Embedded Control Handbook Update 2000

TO OUR VALUED CUSTOMERS:

Welcome to the ***Embedded Control Handbook (ECHB) Update 2000.*** The *ECHB Update* 2000 is the ***third*** in the series of application-orientated publications from Microchip Technology Inc. It includes all new application notes, technical briefs and reference designs which have been written and published since the ***Embedded Control Handbook, Volumes 1 and 2*** were released.

Embedded Control Handbook (ECHB) - Volume 1. The *ECHB - Volume 1* is the first 'Volume' in our library system of PICmicro® 8-Bit microcontroller, Nonvolatile Memory, Secure Data Products, and other product application notes, technical briefs, and reference designs. Volume 1 replaces the 1994/1995 ECHB (released in September 1994) and 1995/1996 ECHB Update I (released in September 1995).

Embedded Control Handbook (ECHB) - Volume 2 Math Library. This book is the second 'Volume' in our library system of product application notes. Volume 2 contains a compilation of fixed-point, floating-point and trigonometry function application notes to help designers use the PICmicro® microcontroller library functions in a C program.

Microchip will continue publishing application notes, technical briefs and reference designs in a series of supplemental handbooks called 'Updates'. Updates will be published annually, providing an uninterrupted flow of current application notes, technical briefs and reference designs for our customers' convenience and use. These Updates, with revised and new documents, will be incorporated into future Volumes as appropriate.

And of course, as individual application notes become available, they will be posted to our web site for download at: **www.microchip.com**.

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your Microchip products. To this end, we will continue to improve our publications to better suit your needs. Our publications will be refined and enhanced as new Volumes and Updates are introduced. We welcome your feedback.

If you have any questions or comments regarding this publication, please contact the Marketing Communications Department via fax at 480.917.4150.

Table of Contents

Table of Contents (continued)

PAGE

Table of Contents (continued)

Update 2000 Subject Index

The following is an alphabetical subject index for the application notes, technical briefs and reference designs that are ONLY available in the *Embedded Control Handbook Update 2000*. For complete listings of other application notes, technical briefs and reference designs available, please refer to "Volume 1 Application Notes" on page xiii, and "Volume 2 Application Notes" on page xvi directly following the subject index.

Update 2000 Subject Index (continued)

Update 2000 Subject Index (continued)

Update 2000 Subject Index (continued)

Volume 1 Application Notes - Alphabetical

The following is an alphabetical list of application notes, technical briefs and reference designs that are available in the Microchip Technology Inc. *Embedded Control Handbook, Volume 1*. Please see your local Microchip Sales Representative, Distributor or Sales Office for the latest copy (order number DS00092).

Volume 1 Application Notes - Alphabetical (continued)

Volume 1 Application Notes - Alphabetical (continued)

Volume 2 Application Notes - Alphabetical

The following is an alphabetical list of application notes that are available in the Microchip Technology Inc. *Embedded Control Handbook, Volume 2, Math Library*. Please see your local Microchip Sales Representative, Distributor or Sales Office for the latest copy (order number DS00167).

Volume 1 Application Notes – Numerical

The following is a numerical list of application notes, technical briefs and reference designs that are available in the Microchip Technology Inc. *Embedded Control Handbook, Volume 1*. Please see your local Microchip Sales Representative, Distributor or Sales Office for the latest copy (order number DS00092).

Volume 1 Application Notes – Numerical (continued)

Volume 1 Application Notes – Numerical (continued)

Volume 2 Application Notes – Numerical

The following is a numerical list of application notes that are available in the Microchip Technology Inc. *Embedded Control Handbook, Volume 2, Math Library*. Please see your local Microchip Sales Representative, Distributor or Sales Office for the latest copy (order number DS00167).

SECTION 1
MICROCHIP TECHNOLOGY INC.
COMPANY PROFILE

Microchip Technology Inc.

Company Profile

The Embedded Control Solutions Company®

Since its inception, Microchip Technology has focused its resources on delivering innovative semiconductor products to the global embedded control marketplace. To do this, we have focused our technology, engineering, manufacturing and marketing resources on synergistic product lines: PICmicro® 8-bit microcontrollers (MCUs), high-endurance Serial EEPROMs, an expanding product portfolio of analog/interface products, RFID tags and KEELOQ® security devices – all aimed at delivering comprehensive, high-value embedded control solutions to a growing base of customers.

Inside Microchip Technology you will find:

- An experienced executive team focused on innovation and committed to listening to our customers
- A focus on providing high-performance, cost-effective embedded control solutions
- Fully integrated manufacturing capabilities
- A global network of manufacturing and customer support facilities
- A unique corporate culture dedicated to continuous improvement
- Distributor network support worldwide including certified distribution FAEs
- A Complete Product Solution including:
  - 8-bit RISC OTP, FLASH, EEPROM and ROM MCUs
  - A full family of advanced analog 8-bit MCUs
  - KEELOQ security devices featuring patented code hopping technology
  - Stand-alone analog and interface products plus microID™ RFID tagging devices
  - A complete line of high-endurance Serial EEPROMs
  - World-class, easy-to-use development tools
  - An Automotive Products Group to engage with key automotive accounts and provide necessary application expertise and customer service

Business Scope

Microchip Technology Inc. designs, manufacturers and markets a variety of CMOS semiconductor components to support the market for cost-effective embedded control solutions.

Microchip's products feature compact size, integrated functionality, ease of development and technical support so essential to timely and cost-effective product development by our customers.

Chandler, Arizona: Company headquarters near Phoenix, Arizona; executive offices, R&D and wafer fabrication occupy this 242,000-square-foot multi-building campus.

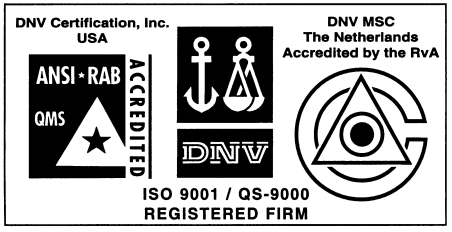
Tempe, Arizona: Microchip's 200,000-square-foot wafer fabrication facility provides increased manufacturing capacity today and for the future.

Market Focus

Microchip targets select markets where our advanced designs, progressive process technology and industry-leading product performance enables us to deliver decidedly superior performance. Our Company is positioned to provide a complete product solution for embedded control applications found throughout the consumer, automotive, telecommunication, office automation and industrial control markets. Microchip products are also meeting the unique design requirements of targeted embedded applications including internet, safety and security.

Certified Quality Systems

Microchip received QS-9000 Quality System certification for its worldwide headquarters and wafer fabrication facilities in September 1999. Microchip's quality system processes and procedures are QS-9000 compliant for all of the Company's devices, including PICmicro 8-bit MCUs, serial EEPROMs, KEELOQ code hopping devices and microperipheral products.



QS-9000 was developed by Chrysler, Ford and General Motors to establish fundamental quality systems that provide for continuous improvement, emphasizing defect prevention and the reduction of variation and waste in the supply chain. Microchip was audited by QS-9000 registrar Det Norske Veritas Certification Inc. of Houston, the same firm which granted Microchip its ISO 9001 Quality System certification in 1997.

QS-9000 certification recognizes Microchip's quality systems conform to the stringent standards set forth by the automotive industry, benefiting all customers.

Fully Integrated Manufacturing

Microchip delivers fast turnaround and consistent quality through total control over all phases of production. Research and development, design, mask making, wafer fabrication, and the major part of assembly and quality assurance testing are conducted at facilities wholly-owned and operated by Microchip. Our integrated approach to manufacturing along with rigorous use of advanced Statistical Process Control (SPC) and a continuous improvement culture has resulted in high and consistent yields which have positioned Microchip as a quality leader in its global markets. Microchip's unique approach to SPC provides customers with excellent pricing, quality, reliability and on-time delivery.

Bangkok, Thailand: Microchip's 200,000-square-foot manufacturing facility houses the technology and assembly/test equipment for high speed testing and packaging.

A Global Network of Plants and Facilities

Microchip is a global competitor providing local services to the world's technology centers. The Company's design and technology advancement facilities, and wafer fabrication sites are located in Chandler and Tempe, Arizona.

The Tempe facility provides an additional 200,000 square feet of manufacturing space that meets the increased production requirements of a growing customer base, and provides production capacity which more than doubles that of Chandler.

Microchip facilities in Bangkok, Thailand, and Shanghai, China, serve as the foundation of Microchip's extensive assembly and test capability located throughout Asia. The use of multiple fabrication, assembly and test sites, with more than 640,000-square-feet of facilities worldwide, ensures Microchip's ability to meet the increased production requirements of a fast growing customer base.

Microchip supports its global customer base from direct sales and engineering offices in Asia, North America, Europe and Japan. Offices are staffed to meet the high quality expectations of our customers, and can be accessed for technical and business support. The Company also franchises more than 60 distributors and a network of technical manufacturer's representatives serving 24 countries worldwide.

Embedded Control Overview

Unlike “processor” applications such as personal computers and workstations, the computing or controlling elements of embedded control applications are embedded inside the application. The consumer is only concerned with the very top-level user interface such as keypads, displays and high-level commands. Very rarely does an end-user know (or care to know) the embedded controller inside (unlike the conscientious PC users, who are intimately familiar not only with the processor type, but also its clock speed, DMA capabilities and so on).

It is, however, most vital for designers of embedded control products to select the most suitable controller and companion devices. Embedded control products are found in all market segments: consumer, commercial, PC peripherals, telecommunications, automotive and industrial. Most embedded control products must meet special requirements: cost effectiveness, low-power, small-footprint and a high level of system integration.

Typically, most embedded control systems are designed around a MCU which integrates on-chip program memory, data memory (RAM) and various peripheral functions, such as timers and serial communication. In addition, these systems usually require complementary Serial EEPROM, analog/interface devices, display drivers, keypads or small displays.

Microchip has established itself as a leading supplier of embedded control solutions. The combination of high-performance PIC12CXXX, PIC16C5X, PIC16CXXX, PIC17CXXX and PIC18CXXX MCU families with Migratable Memory™ technology, along with non-volatile memory products, provide the basis for this leadership. By further expanding our product portfolio to provide precision analog and interface products, Microchip is committed to continuous innovation and improvement in design, manufacturing and technical support to provide the best possible embedded control solutions to you.

PICmicro MCU Overview and Roadmap

Microchip PICmicro MCUs combine high-performance, low-cost, and small package size, offering the best price/performance ratio in the industry. More than 900 million of these devices have shipped to customers worldwide since 1990. Microchip offers five families of 8-bit MCUs to best fit your application needs:

- PIC12CXXX 8-pin 12-bit/14-bit program word
- PIC16C5X 12-bit program word
- PIC16CXXX 14-bit program word
- PIC17CXXX 16-bit program word
- PIC18CXXX enhanced 16-bit program word

All families offer OTP, low-voltage and low-power options, with a variety of package options. Selected members are available in ROM, EEPROM or reprogrammable FLASH versions.

PIC12CXXX: 8-Pin, 8-Bit Family

The PIC12CXXX family packs Microchip’s powerful RISC-based PICmicro architecture into 8-pin DIP and SOIC packages. These PIC12CXXX products are available with either a 12-bit or 14-bit wide instruction set, a low operating voltage of 2.5V, small package footprints, interrupt handling, a deeper hardware stack, multiple channels and EEPROM data memory. All of these features provide an intelligence level not previously available in applications because of cost or size considerations.

PIC16C5X: 12-Bit Architecture Family

The PIC16C5X is the well-established base-line family that offers the most cost-effective solution. These PIC16C5X products have a 12-bit wide instruction set and are currently offered in 14-, 18-, 20- and 28-pin packages. In the SOIC and SSOP packaging options, these devices are among the smallest footprint MCUs in the industry. Low-voltage operation, down to 2.0V for OTP MCUs, makes this family ideal for battery operated applications. Additionally, the PIC16HV5XX can operate up to 15 volts for use directly with a battery.

PIC16CXXX: 14-Bit Architecture Family

With the introduction of new PIC16CXXX family members, Microchip now provides the industry’s highest performance Analog-to-Digital Converter capability at 12-bits for an 8-bit MCU. The PIC16CXXX family offers a wide-range of options, from 18- to 68-pin packages as well as low to high levels of peripheral integration. This family has a 14-bit wide instruction set, interrupt handling capability and a deep, 8-level hardware stack. The PIC16CXXX family provides the performance and versatility to meet the more demanding requirements of today’s cost-sensitive marketplace for mid-range 8-bit applications.

PIC17CXXX: 16-Bit Architecture Family

The PIC17CXXX family offers the world’s fastest execution performance of any 8-bit MCU family in the industry. The PIC17CXXX family extends the PICmicro MCU’s high-performance RISC architecture with a 16-bit instruction word, enhanced instruction set and powerful vectored interrupt handling capabilities. A powerful array of precise on-chip peripheral features provides the performance for the most demanding 8-bit applications.

PIC18CXXX: 16-Bit Enhanced Architecture Family

The PIC18CXXX is a family of high performance, CMOS, fully static, 16-bit MCUs with integrated analog-to-digital (A/D) converter. All PIC18CXXX MCUs incorporate an advanced RISC architecture. The PIC18CXXX has enhanced core features, 32 level-deep stack, and multiple internal and external interrupts sources. The separate instruction and data busses of the Harvard architecture allow a 16-bit wide instruction word with the separate 8-bit wide data. The two-stage instruction pipeline allows all instructions to execute in a single cycle, except for program branches, which require two cycles. A total of 77 instructions (reduced instruction set) are available. Additionally, a large register set gives some of the architectural

innovations used to achieve a very high performance of 10MIPS for an 8-bit MCU. The PIC18CXXX family has special features to reduce external components, thus reducing cost, enhancing system reliability and reducing power consumption. These include programmable Low Voltage Detect (LVD) and programmable Brown-Out Detect (BOD).

The Mechatronics Revolution

We are living through a revolutionary period that is impacting almost every aspect of our lives. The nature of the revolution is the momentous shift from analog/electro-mechanical timing and control to digital electronics. It is called the Mechatronics Revolution, and it is being staged in companies throughout the world, with design engineers right on the front lines: make it smarter, make it smaller, make it do more, make it cost less to manufacture – and make it snappy.

To meet the needs of this growing customer base, Microchip is rapidly expanding its already broad line of 8-bit PICmicro MCUs. The PIC12CXXX family's size opens up new possibilities for product design.

PICmicro MCU Naming Convention

The PICmicro architecture offers users a wider range of cost/performance options than any 8-bit MCU family. In order to identify the families, the following naming conventions have been applied to the PICmicro MCUs:

TABLE 1: PICmicro MCU NAMING CONVENTION*

Family		Architectural Features	Name	Technology
PIC18CXXX	8-bit Higher-Performance MCU Family	• 10 MIPS @ 40 MHz • 4x PLL clock • 16-bit wide instruction set • C compiler efficient instruction set • Internal/external vectored interrupts	PIC18Cxx2	OTP program memory with higher resolution analog functions
PIC17CXXX	8-bit High-Performance MCU Family	• 16-bit wide instruction set • Internal/external vectored interrupts • DC - 33 MHz clock speed • 120 ns instruction cycle (@ 33 MHz) • Hardware multiply	PIC17C4X	OTP program memory, digital only
			PIC17CR4X	ROM program memory, digital only
			PIC17C7XX	OTP program memory with mixed-signal functions
PIC16CXXX	8-bit Mid-Range MCU Family	• 14-bit wide instruction set • Internal/external interrupts • DC - 20 MHz clock speed (Note 1) • 200 ns instruction cycle (@ 20 MHz)	PIC14CXXX	OTP program memory with A/D and D/A functions
			PIC16C55X	OTP program memory, digital only
			PIC16C6X	OTP program memory, digital only
			PIC16CR6X	ROM program memory, digital only
			PIC16C62X	OTP program memory with comparators
			PIC16CR62X	ROM program memory with comparators
			PIC16CE62X	OTP program memory with comparators and EEPROM data memory
			PIC16F62X	FLASH program memory with comparators and EEPROM data memory
			PIC16C64X	OTP program memory with comparators
			PIC16C66X	OTP program memory with comparators
			PIC16C7X	OTP program memory with analog functions (i.e. A/D)
			PIC16CR7X	ROM program memory with analog functions
			PIC16C7XX	OTP program memory with higher resolution analog functions
			PIC16F8X	FLASH program memory and EEPROM data memory
			PIC16CR8X	ROM program memory and EEPROM data memory
			PIC16F87X	FLASH program memory with higher resolution analog functions
			PIC16C9XX	OTP program memory, LCD driver
PIC16C5X	8-bit Base-Line MCU Family	• 12-bit wide instruction set • DC - 20 MHz clock speed • 200 ns instruction cycle (@ 20 MHz)	PIC16C5X	OTP program memory, digital only
			PIC16CR5X	ROM program memory, digital only
			PIC16C505	OTP program memory, digital only, internal 4MHz oscillator
			PIC16HV540	OTP program memory with high voltage operation
PIC12CXXX	8-bit, 8-pin MCU Family	• 12- or 14-bit wide instruction set • DC - 10 MHz clock speed • 400 ns instruction cycle (@ 10 MHz) • Internal 4MHz oscillator	PIC12C5XX	OTP program memory, digital only
			PIC12CE5XX	OTP program memory, digital only with EEPROM data memory
			PIC12CR5XX	ROM program memory, digital only
			PIC12C67X	OTP program memory with analog functions
			PIC12CE67X	OTP program memory with analog functions and EEPROM data memory

Note 1: The maximum clock speed for some devices is less than 20 MHz.

*Please check with your local Microchip distributor, sales representative or sales office for the latest product information.

Development Systems

Microchip is committed to providing useful and innovative solutions to your embedded system designs. Our installed base of application development systems has grown to an impressive 150,000 systems worldwide.

Among support products offered are MPLAB™-ICE 2000 In-Circuit Emulator running under the Windows® environment. This new real-time emulator supports low-voltage emulation, to 2.0 volts, and full-speed emulation. MPLAB, a complete Integrated Development Environment (IDE), is provided with MPLAB-ICE 2000. MPLAB allows the user to edit, compile and emulate from a single user interface, making the developer productive very quickly. MPLAB-ICE 2000 is designed to provide product development engineers with an optimized design tool for developing target applications. This universal in-circuit emulator provides a complete MCU design toolset for PICmicro MCUs in the PIC12CXXX, PIC16C5X, PIC16CXXX, PIC17CXXX and PIC18CXXX families. MPLAB-ICE 2000 is CE compliant.

Microchip's newest development tool, MPLAB In-Circuit Debugger (ICD) Evaluation Kit, uses the in-circuit debugging capabilities of the PIC16F87X MCU family and Microchip's ICSP™ capability to debug source code in the application, debug hardware in real time and program a target PIC16F87X device.

PRO MATE® II, the full-featured, modular device programmer, enables you to quickly and easily program user software into PICmicro MCUs, HCS products and Serial EEPROMs. PRO MATE II runs under MPLAB IDE and operates as a stand-alone unit or in conjunction with a PC-compatible host system.

The PICSTART® Plus development kit, is a low-cost development system for the PIC12CXXX, PIC16C5X, PIC16CXXX and PIC17CXXX MCUs.

PICDEM low-cost demonstration boards are simple boards which demonstrate the basic capabilities of the full range of Microchip's MCUs. Users can program the sample MCUs provided with PICDEM boards, on a PRO MATE II or PICSTART Plus programmer, and easily test firmware. KEELOQ Evaluation Tools support Microchip's HCS Secure Data Products.

The Serial EEPROM Designer's Kit includes everything necessary to read, write, erase or program special features of any Microchip Serial EEPROMs. The *Total Endurance*™ Disk is included to aid in trade-off analysis and reliability calculations. The total kit can significantly reduce time-to-market and result in an optimized system.

TABLE 2: PICmicro SYNERGISTIC DEVELOPMENT TOOLS

Development Tool	Name	PIC12CXXX	PIC16C5X	PIC16CXXX	PIC16F87X	PIC17CXXX	PIC18CXXX
Integrated Development Environment (IDE)	MPLAB™	✔	✔	✔	—	✔	✔
C Compiler	MPLAB-C17	—	—	—	—	✔	
C Compiler	MPLAB-C18	—	—	—	—	—	✔
Full-Featured, Modular In-Circuit Emulator	MPLAB-ICE 2000	✔	✔	✔	—	✔	✔
In-Circuit Debugger Evaluation Kit	MPLAB-ICD	—	—	—	✔	—	—
Full-Featured, Modular Device Programmer	PRO MATE® II	✔	✔	✔	—	✔	✔
Entry-Level Development Kit with Programmer	PICSTART® Plus	✔	✔	✔	—	✔	✔

Software Support

MPLAB Integrated Development Environment (IDE) is a Windows-based development platform for Microchip's PICmicro MCUs. MPLAB IDE offers a project manager and program text editor, a user-configurable toolbar containing four pre-defined sets and a status bar which communicates editing and debugging information.

MPLAB-IDE is the common user interface for Microchip development systems tools including MPLAB Editor, MPASM Assembler, MPLAB-SIM Software Simulator, MPLIB, MPLINK, MPLAB-C17 Compiler, MPLAB-C18 Compiler, MPLAB-ICE 2000, PRO MATE II Programmer and PICSTART Plus Development Programmer.

Microchip endeavors at all times to provide the best service and responsiveness possible to its customers. The Microchip Internet site can provide you with the latest technical information, production released software for development tools, application notes and promotional news on Microchip products and technology. The Microchip World Wide Web address is http://www.microchip.com.

Secure Data Products Overview

Microchip's patented KEELOQ® code hopping technology is the perfect solution for remote keyless entry and logical/physical access control systems. The initial device in the family, HCS300 encoder, replaces current fixed code encoders in transmitter applications providing a low cost, integrated solution. The KEELOQ family is continuing to expand with the HCS301 (high voltage encoder), HCS200 (low-end, low-cost encoder), and high-end encoders (HCS360 and HCS361) that meet OEM specifications and requirements. The HCS410, a self-powered transponder superset of the HCS360, is the initial device in a new and expanding encoder/transponder family.

Microchip provides flexible decoder solutions by providing optimized routines for Microchip's PICmicro MCUs. This allows the designer to combine the decoder and system functionality in a MCU. The decoder routines are available under a license agreement. The HCS500, HCS512 and HCS515 are the first decoder devices in the KEELOQ family. These devices are single chip decoder solutions and simplify designs by handling learning and decoding of transmitters.

The KEELOQ product family is expanding to include enhanced encoders and decoders. Typical applications include automotive RKE, alarm and immobilizer systems, garage door openers and home security systems.

KEELOQ® Rock Solid Security

KEELOQ Encoder Devices					
Product	Transmission Code Length Bits	Code Hopping Bits	Prog. Encryption Key Bits	Seed Length	Operating Voltage
HCS101*	66	—	—	—	3.5V to 13.0V
HCS200	66	32	64	32	3.5V to 13.0V
HCS201*	66	32	64	32	3.5V to 13.0V
HCS300	66	32	64	32	2.0V to 6.3V
HCS301	66	32	64	32	3.5V to 13.0V
HCS320	66	32	64	32	3.5V to 13.0V
HCS360	67	32	64	48	2.0V to 6.6V
HCS361	67	32	64	48	2.0V to 6.6V
HCS365*	69	32	2 x 64	60	2.0V to 6.6V
HCS370*	69	32	2 x 64	60	2.0V to 6.6V
HCS410	69	32	64	60	2.0V to 6.6V
HCS412*	69	32	64	60	2.0V to 6.6V
HCS470*	69	32	2 x 64	60	2.0V to 6.6V
KEELOQ Decoder Devices					
Product	Reception Length Bits	Transmitters Supported	Functions		Operating Voltage
HCS500	67	Up to 7	15 Serial Functions		4.5V to 5.5V
HCS512	67	Up to 4	15 (S0, S1, S2, S3); VLOW, Serial		3.0V to 6.0V
HCS515	67	Up to 7	15 Serial; 3 Parallel		4.5V to 5.5V

*Contact Microchip Technology Inc. for availability.

Analog/Interface Products

Using its technology achievements in developing analog circuity for its PICmicro MCU family, the Company launched a complementary line of stand-alone analog and interface products. Many of these stand-alone devices support functionality that may not currently available on PICmicro MCUs. Stand-alone analog IC products currently offered include:

- Analog-to-Digital Converters
- Operational Amplifiers
- System Supervisors

Microchip also offers innovative silicon products to support a variety of bus interfaces used to transmit data to and from embedded control systems. The first interface products support Controller Area Network (CAN), a bus protocol highly integrated into a variety of networked applications including automotive.

High-Performance 12-Bit Analog-to-Digital Converters

The MCP320X 12-bit analog-to-digital converter (ADC) family is based on a successive approximation register architecture. The first four members include: MCP3201, MCP3202, MCP3204 and MCP3208. The MCP320X family features 100K samples per second throughput, low power of 400 microamps active and 500 nanoamps standby, wide supply voltage of 2.7-5.5 volts, extended industrial temperature range of –40º to 85º, +/- 1 LSB DNL and +/- 1 LSB INL max. at 100 ksps., guaranteed no missing codes, and a serial output with an industry-standard SPI® bus interface. The MCP320X is available in 1-, 2-, 4-, and 8-input channel versions (the MCP3201, MPC3202, MCP3204 and MCP3208, respectively). The devices are offered

in PDIP, SOIC and TSSOP packages. Applications include data acquisition, instrumentation and measurement, multi-channel data loggers, industrial PCs, motor control, robotics, industrial automation, smart sensors, portable instrumentation, and home medical appliances.

Controller Area Network (CAN)

Microchip is enhancing its product portfolio by introducing the CAN Product Family. The MCP2510 is the smallest, easiest-to-use, CAN controller on the market today. Combining the MCP2510 with Microchip's broad range of high-performance PICmicro MCUs enables Microchip to support for virtually all of today's CAN-based applications. Other potential benefits of having a separate CAN controller include the ability for system designers to select from a much wider variety of MCUs for an optimal performance solution.

Additional products planned for Microchip's CAN product portfolio include other CAN peripherals and a family of PICmicro MCUs with integrated CAN support. Future support for CAN bus applications includes a family of PICmicro MCUs and additional CAN peripheral devices.

Operational Amplifiers

The Microchip MCP60X Operational Amplifier family includes four devices: MCP601, MCP602, MCP603 and MCP604. These devices are Microchip's first 2.7 volt single supply operational amplifier products. The MCP60X family offers a gain bandwidth product of 2.8MHz with low typical operating current of 230µA. The MCP60X devices use Microchip's advanced CMOS technology which provides low bias current, high speed operation, high open-loop gain and rail-to-rail output swing.

System Supervisors

Microchip offers a complete family of system supervisor products. The new devices include the MCP809/810 and MCP100/101 supervisory circuits with push-pull output and the MCP120/130 supervisory circuits with open drain output. The devices are functionally and pin-out comparable to products from other analog suppliers.

microID™ RFID Tagging Devices

Only Microchip manufactures world-class components for every application in the radio frequency identification (RFID) system. From the advanced, feature-packed microID family of RFID tags and high-endurance Serial EEPROMs to high performance PICmicro MCUs and KEELOQ code hopping encoders - Microchip's full range of RFID solutions are available for your tag, peripheral and reader application designs.

The microID family can emulate almost any standard on the market today. It provides drop-in compatible solutions to the most commonly used 125kHz and 13.56 MHz tags and an upgrade migration path for virtually any application with higher performance and new features.

Serial EEPROM Overview

Microchip offers one of the broadest selections of CMOS Serial EEPROMs on the market for embedded control systems. Serial EEPROMs are available in a variety of densities, operating voltages, bus interface protocols, operating temperature ranges and space saving packages.

Densities:

Currently range from 128 bits to 256K bits with higher density devices in development.

Bus Interface Protocols:

We offer all popular protocols: I^2C™, Microwire® and SPI.

Operating Voltages:

In addition to standard 5V devices there are two low voltage families. The "LC" devices operate down to 2.5V, while the breakthrough "AA" family operates, in both read and write mode, down to 1.8V, making these devices highly suitable for alkaline and NiCd battery powered applications.

Temperature Ranges:

Like all Microchip devices, many Serial EEPROMs are offered in Commercial (0°C to +70°C), Industrial (-40°C to +85°C) and Extended (-40°C to +125°C) operating temperature ranges.

Packages:

Small footprint packages include: industry standard 5-lead SOT-23, 8-lead DIP, 8-lead SOIC in JEDEC and EIAJ body widths, and 14-lead SOIC. The SOIC comes in two body widths; 150 mil and 207 mil.

Technology Leadership:

Selected Microchip Serial EEPROMs are backed by a 1 million Erase/Write cycle guarantee. Microchip's erase/write cycle endurance is among the best in the world, and only Microchip offers such unique and powerful development tools as the Total Endurance disk. This mathematical software model is an innovative tool used by system designers to optimize Serial EEPROM performance and reliability within the application.

Microchip offers Plug-and-Play to the DIMM module market with the 24LCS52, a special function single-chip EEPROM that is available in space saving packages. For Plug-and-Play video monitor applications, Microchip offers the 24LC21, a single-chip DDC1™/DDC2™-compatible solution. In addition, Microchip released a high-speed 1 MHz 2-wire Serial EEPROM device ideal for high-performance embedded systems.

Microchip is a high-volume supplier of Serial EEPROMs to all the major markets worldwide. The Company continues to develop new Serial EEPROM solutions for embedded control applications.

OTP EPROM Overview

Microchip's CMOS EPROM devices are produced in densities from 64K to 512K. Typical applications include computer peripherals, instrumentation, and automotive devices. Microchip's expertise in surface mount packaging on SOIC and TSOP packages led to the development of the surface mount OTP EPROM market where Microchip is a leading supplier today. Microchip is also a leading supplier of low-voltage EPROMs for battery powered applications.

MIGRATABLE MEMORY™ TECHNOLOGY

Microchip's innovative Migratable Memory technology (MMT) provides socket and software compatibility among all of its equivalent ROM, one-time-programmable (OTP) and FLASH memory MCUs. MMT allows customers to match the selection of MCU memory technology to the product life cycle of their application, providing an easy migration path to a lower cost solution whenever appropriate.

FLASH memory is an ideal solution for engineers designing products for embedded systems – especially during the development and early stages of the product. In certain products and applications, FLASH memory may be used for the life of the product because of the advantages of field upgradability or where product inventory flexibility is required.

Once the design enters the pre-production stage and continues through introduction and growth stages, OTP program memory provides maximum programming flexibility and minimum inventory scrappage. The OTP device is pin and socket compatible with the FLASH device – providing a lower cost, high-volume flexible solution.

As the design enters a mature stage and program code stabilizes, a lower cost, socket compatible ROM memory device could be used. In some cases, OTP memory may still be used as the most cost-effective memory technology for the product. Compatibility and flexibility are key to the success of the PICmicro MCU product family, and ultimately the success of our customers.

FLEXIBLE PROGRAMMING OPTIONS

To meet the stringent design requirements placed on our customers, the following innovative programming options are offered. These programming options address procurement issues by reducing and limiting work-in-process liability and facilitating finished goods code revisions. Microchip's worldwide distributors stock reprogrammable and one-time programmable inventory, allowing customers to respond to immediate sales opportunities or accommodate engineering changes off the shelf.

FLASH (electrically reprogrammable)

PICmicro FLASH MCUs allow erase and reprogramming of the MCU program memory. Reprogrammability offers a highly flexible solution to today's ever-changing market demands – and can substantially reduce time to market. Users can program their systems very late in the manufacturing process or update systems in the field. This allows easy code revisions, system parameterization or customer-specific options with no scrappage. Reprogrammability also reduces the design verification cycle.

One-Time Programmable (OTP)

PICmicro OTP MCUs are manufactured in high volumes without customer specific software and can be shipped immediately for custom programming. This is useful for customers who need rapid time to market and flexibility for frequent software updates.

In-Circuit Serial Programming™ (ICSP™)

Microchip's PICmicro FLASH and OTP MCUs feature ICSP capability. ICSP allows the MCU to be programmed after being placed in a circuit board, offering tremendous flexibility, reduced development time, increased manufacturing efficiency and improved time to market. This popular technology also enables reduced cost of field upgrades, system calibration during manufacturing, the addition of unique identification codes to the system and system calibration. Requiring only two I/O pins for most devices, Microchip offers the most non-intrusive programming methodology in the industry.

Self Programming

Microchip's PIC16F87X family features self programming capability. Self programming enables remote upgrades to the FLASH program memory and the end equipment through a variety of medium ranging from Internet and Modem to RF and Infrared. To setup for self programming, the designer programs a simple boot loader algorithm in a code protected area of the FLASH program memory. Through the selected medium, a secure command allows entry into the PIC16F87X MCU through the USART, I^2C or SPI serial communication ports. The boot loader is then enabled to reprogram the PIC16F87X FLASH program memory with data received over the desired medium. And, of course, self programming is accomplished without the need for external components and without limitations on the PIC16F87X's operating speed or voltage.

Quick-Turn Programming (QTP)

Microchip offers a QTP programming service for factory production orders. This service is ideal for customers who choose not to program a medium to high unit volume in their own factories, and whose production code patterns have stabilized.

Serialized Quick-Turn Programming (SQTP^{SM})

SQTP is a unique, flexible programming option that allows Microchip to program serialized, random or pseudo-random numbers into each device. Serial programming allows each device to have a unique number which can serve as an entry-code, password or ID number.

Masked ROM

Microchip offers Masked ROM versions of many of its most popular PICmicro MCUs, giving customers the lowest cost option for high volume products with stable firmware.

Future Products and Technology

Microchip is constantly developing advanced process technology modules and new products that utilize our advanced manufacturing capabilities. Current production technology utilizes lithography dimensions down to 0.7 micron.

Microchip's research and development activities include exploring new process technologies and products that have industry leadership potential. Particular emphasis is placed on products that can be put to work in high-performance broad-based markets.

Equipment is continually updated to bring the most sophisticated process, CAD and testing tools online. Cycle times for new technology development are continuously reduced by using in-house mask generation, a high-speed pilot line within the manufacturing facility and continuously improving methodologies.

Objective specifications for new products are developed by listening to our customers and by close co-operation with our many customer-partners worldwide.

NOTES:

SECTION 2
PICmicro® 8-BIT MICROCONTROLLER APPLICATION NOTES AND TECHNICAL BRIEFS

Engineer's Assistant Using a PIC16F84A

Author: *Voja Antonic*
PC Press

INTRODUCTION

This compact instrument is intended to be a digital laboratory tool for hardware and, in some cases, software debugging. It contains four instruments in one unit: logic probe, single channel logic state analyzer, frequency counter and serial code receiver.

The only chip used is a PIC16F84A running at 10 MHz. The display unit is a LCD dot matrix alphanumeric module with 2 rows of 20 characters. The LCD is used as the display device for all functions, except for the logic probe which indicates low, high and pulse logic states on individual LEDs. Mode select, parameter change, function execute and ON/OFF switching is activated by two keys.

The probe tip is the common input for all functions, and the GND cable is used for connection to Vss of the tested circuit.

Although there are a lot of functions integrated in a single chip unit, it did not increase the complexity of hardware, as all functions are implemented in software. This enables a very good price/performance ratio.

The power supply is obtained by four 1.2V/180 mAh or 250 mAh NiCd batteries of LR03 (AAA) size. The instrument also has a battery manager, which supports automatic battery discharging and charging.

The source code is written in MPASM. As it is highly optimized for code space, most of the code could not be written in a modular format. For the same reason, a lot of subroutines have more than one entry point and some of them are terminated by a `GOTO` instruction instead of using a `RETURN` instruction.

FEATURES

- Stand-alone hand-held instrument
- Single chip design
- Built-in rechargeable power supply
- Easy to assemble and ready to use, no adjustment needed
- User interface with LCD output and command input by two keys
- TTL or 5V CMOS input, or direct input from RS-232C +/-12V signals

SPECIFIC FEATURES FOR INDIVIDUAL FUNCTIONS

Logic Probe

The low and high logic levels are displayed by LEDs, which are OFF if the probe tip is floating or connected to a hi-impedance (>220k) output. A pulse transition is detected and is indicated by turning on the LED for 80 ms.

Logic State Analyzer

The analyzer fetches 300 single bit samples at a selectable rate (in 16 steps from 40 Hz to 1 MHz). It has a programmable start at High-to-Low or Low-to-High transition at input. Digital waveforms are displayed in a pseudographic mode on the LCD.

Serial Code Receiver

The Serial Code Receiver receives 42 bytes and displays them in both HEX and ASCII. The baud rate is selectable in 8 steps, from 1200 to 115200. The selectable format is 7 or 8 bits with or without a parity bit which is not displayed. Signal polarity is also selectable. Direct signal stealing from an RS-232 or an RS-232C interface is possible.

Frequency Counter

The Frequency Counter counts frequency and displays it in an 8-digit decimal format on the LCD with a refresh rate of 500 ms. There are four ranges, from 5 to 40 MHz, which affect the count resolution (from 4 to 32).

Battery Manager

The Battery Manager provides for discharging with an automatic switch that changes to charge mode at 4V battery voltage, charging with 18 mA of constant current and automatic power off after 14 hours. Any DC source between 10V and 30V, at any polarity, can be used for charging.

SYSTEM FUNCTIONS

User Interface

There are four modes of operation: Analyzer, Serial Code Receiver, Frequency Counter and Battery Manager. The logic probe function is transparent in all modes except in the Frequency Counter.

In all modes, submodes are listed in the lower row of the LCD. The submodes list can be cycled through by pressing the right key, which moves the cursor (blinking block) to the right. The left key activates the selected submode (executes a function or changes the parameter state/value).

The right-most submode (right arrow symbol) acts as a shortcut jump to the next mode. After power-on (by pressing any key), a mode is chosen by pressing the left key, then the submode by the right key, and then the eventual parameter change or command execution by the left key again. The only exception is the Logic Probe function, the only action needed is to switch the instrument on, and the logic probe is ready to use.

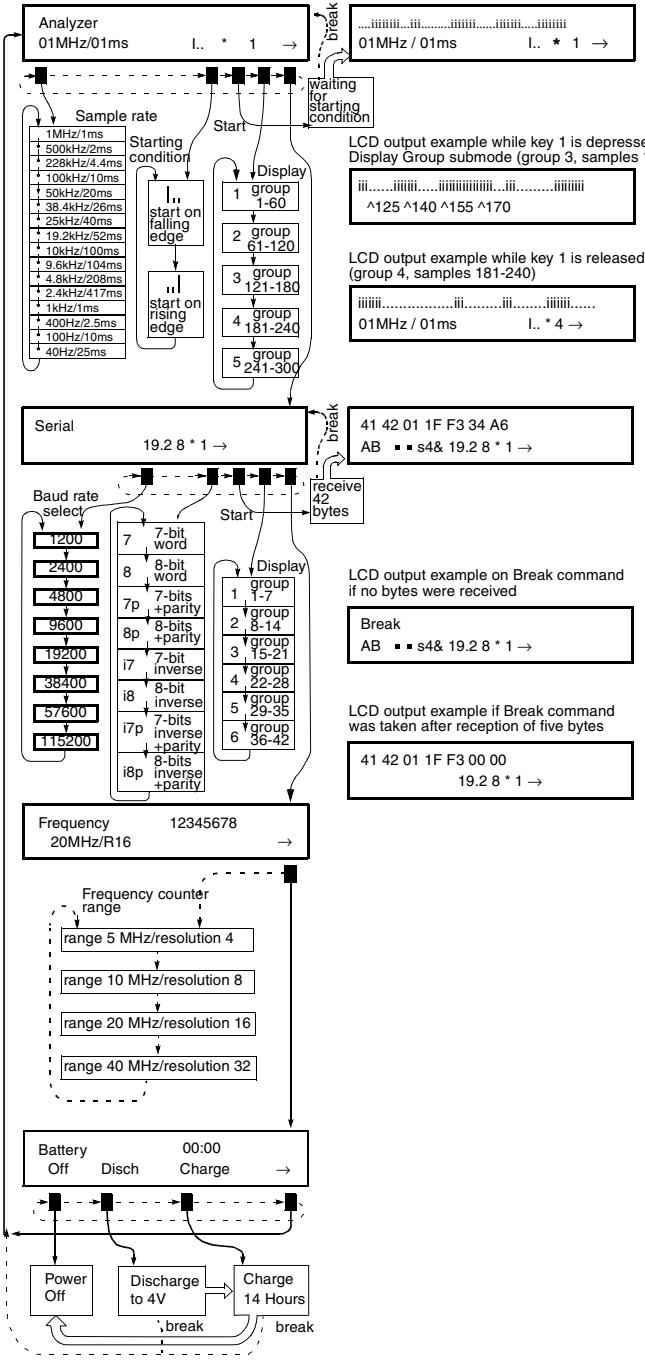
In Analyzer and Serial Code Receiver mode, the asterisk (*) is a special symbol for the "Start" command. When executed (left key pressed while the cursor is on asterisk), it causes the program to wait for a start condition or a start bit.

Although there is a manual Switch Off command (accessible in Battery mode), there is also the automatic power off after approximately 8 minutes of inactivity if no key is pressed. Note that the down counter for automatic power-off is "frozen", while the instrument is waiting for a start condition in analyzer mode and for the start bit in serial code receiver mode. Of course, the same applies to the discharging and charging processes, as another conditions are used to define the end of those processes.

Figure 1 represents the key functions diagram. The dotted line represents actions taken when the right key is pressed, and the solid line is for the left key. The cursor, which is the blinking block on the LCD, is represented as a solid block in Figure 1, but it is moved down on the drawing for clarity.

A variable (named `REL`) in the assembler source code, defines the position of the cursor on the LCD. If it is a '1' (default), the cursor will be placed on the first character of the command (or parameter). If it is a '0', the code will be assembled so that the cursor will be moved to the preceding location (if it exists) before the command.

FIGURE 1: HIGH LEVEL FUNCTION FLOWCHART



Logic Probe

The typical hardware solution for a logic probe is shown in Figure 2. Two inverters, for low and high indication, and two monostables, for pulse detection, are commonly used in most low-cost logic probes. This solution will display an unconnected probe tip as high logic level. There are some better versions which can detect a floating input and turn all LEDS off if it is detected. Figure 3 represents the common solution for such functions, where two analog comparators are employed to detect the low, high and floating inputs.

FIGURE 2: TYPICAL LOGIC PROBE SCHEMATIC

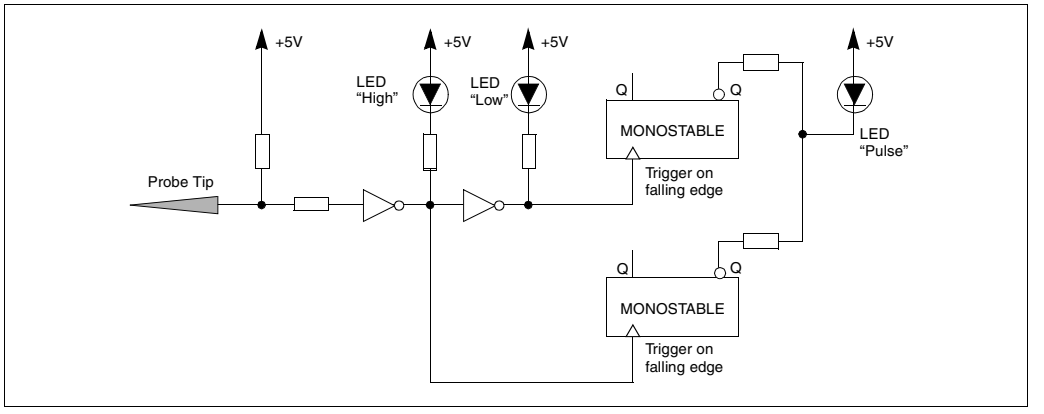
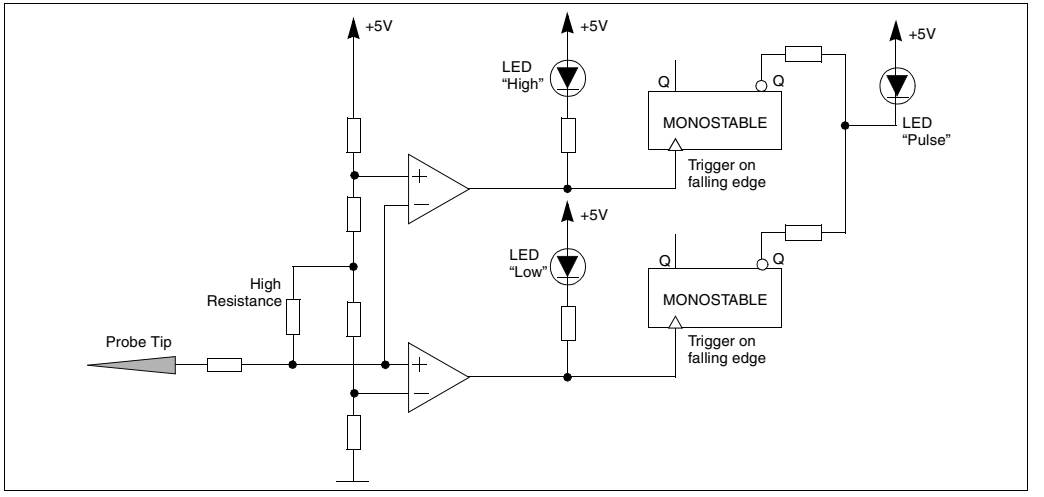


FIGURE 3: IMPROVED LOGIC PROBE SCHEMATIC



Instead of using such approaches, the logic probe function in this instrument is software aided, and the floating input is detected in a dynamic way instead of a static one. The equivalent hardware schematic diagram of this solution is shown in Figure 4 (Pulse detection circuit not shown). The hardware detail which supports the operation of the logic probe used in this unit is represented by Figure 5. The microcontroller polls the input tip and services LEDs L and H. If a transition is detected, LED P is switched ON and the down counter switches it OFF after 80 ms if no additional transition is detected.

This approach has two disadvantages. Logic state latching at a uniform rate may cause visible interference if the frequency of the monitored signal is near the latching rate. This problem is minimized by adding a self-variable extra delay in software, which makes the latching frequency unstable. This makes the range of critical frequencies much wider, but the interference appears as a very short burst of pauses in LED L or H activity, which is completely avoided by adding an extra debouncer of only 250 microseconds. Although unnoticeable, this delay helps prevent LED level instability while monitoring critical frequencies.

FIGURE 4: FUNCTIONAL SCHEMATIC OF PIC16F84A LOGIC PROBE

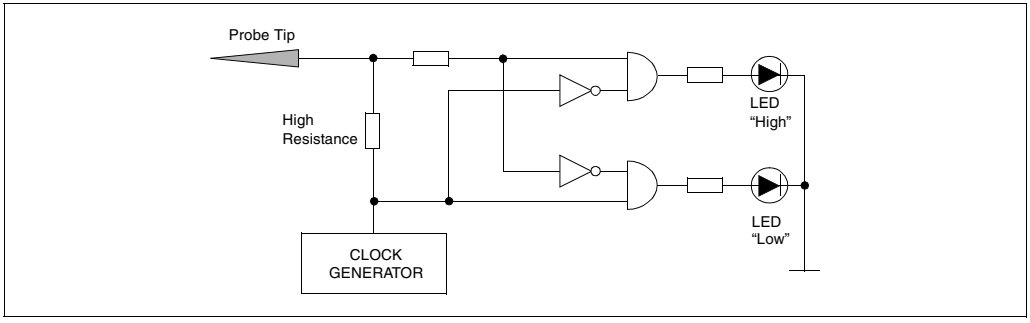
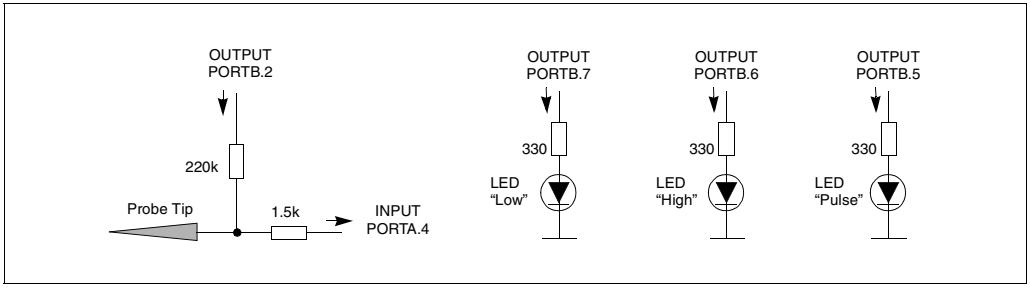


FIGURE 5: SUPPORTING PIC16F84A LOGIC PROBE CIRCUITS



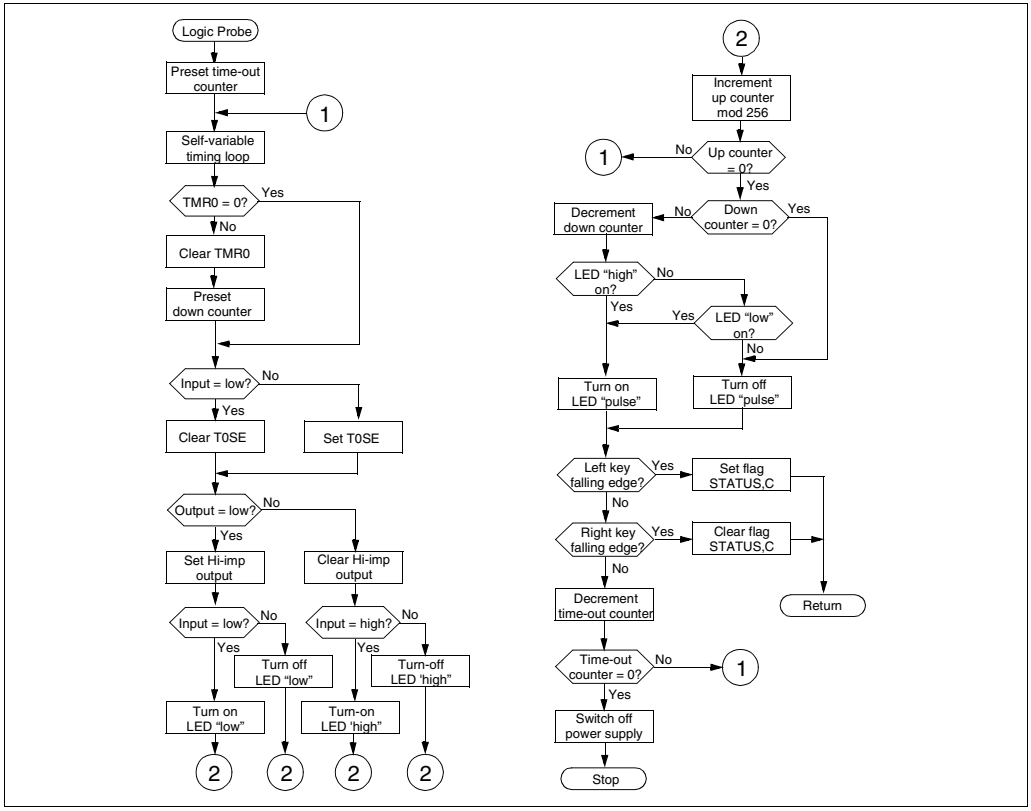
Another disadvantage is related to pulse indication on LED P. In the case of a very short pulse, it is likely that the microcontroller, which polls the input, may omit it between two input reads. Instead of simple polling, the internal counter, TMR0, is used here so that instead of testing the logic state of the input, the state of TMR0 is tested. In this way, pulses as short as 10 ns might be detected. In reality, the minimal pulse width is limited by resistor R6 and the input pin RA4 capacitance. The T0SE bit in the OPTION_REG register is properly updated at each pass, so that the first incoming transition will increment TMR0.

The logic probe software support is integrated in the keyboard routine. LEDs L, H and P are active only while the instrument is idle (doing nothing but waiting for some key to be pressed), which is all the time while the unit is ON, except in frequency counter mode, during battery discharging or charging, or if the START command is issued in analyzer or serial receiver mode and the job (300 samples fetched or all bytes received) is not yet finished.

Pin RB2 is the output which generates square-wave pulses. These pulses are fed through R5 to the probe tip. The resistance is high enough not to affect the tested circuit, except if the tested point is the floating input. However, in that case it will probably make the circuit unstable and thus help in locating the floating input. This pulse stream is also used by software to detect the floating probe tip, and in this case to switch all LEDS off. This saves energy in batteries and helps to detect if the probe tip is validly connected to the point under test.

A simplified flow chart for the logic probe is represented by Figure 6. As this subroutine is an integral part of the key scan routine, the key (debouncers are not shown in detail) and time-out testing (which employs a 16-bit counter, "Time-out Counter") are also provided. "Up Counter" is the free running counter which enables execution of the second part of the subroutine to be performed at each 256th pass. "Down Counter" is the timing base for the LED Pulse. If the state of this counter is greater than zero and the LED Low or LED High is on, the LED Pulse will be turned on. The program exits only if some key is pressed (flag bit STATUS,C denotes which) or when the time-out counter reaches zero.

FIGURE 6: LOGIC PROBE FLOWCHART



Logic State Analyzer

The commonly used hardware concept for a logic analyzer design is represented in Figure 7. All those functions are realized in software, which is much easier to implement, but results in a loss of sampling speed. The software solution is briefly represented on the flow chart in Figure 8.

In analyzer mode, a sequence of 300 one-bit fetches is performed. Samples are stored in internal RAM (actually, 304 samples are read, but the last 4 are dummy reads). The upper row of the LCD is used to display the samples. As the LCD (Hitachi's LM032L) has no graphic capabilities (it is not possible to address a single dot), this is simulated by eight special user-defined characters (which are stored in the character generator RAM), each for a group of 3-bit samples. This enables a pseudo-graphic mode which, in this case, looks as if all pixels were individually addressed.

The display shows a window of 60 samples. One of five windows is selected by placing the cursor on the group number and advancing it by pressing the left key. While the key is pressed, the lower row displays the numeric pointers, which help by counting the sample number and calculating the timings in the recorded sequence. When the key is released, the normal row 2 is restored.

A uniform clock, for sample rate, is internally generated. It is selectable to 16 steps. The frequency and period are both displayed. The following is a list of available sample rates:

1 MHz	50 kHz	10 kHz	1 kHz
500 kHz	38.4 kHz	9.6 kHz	400 Hz
228 kHz	25 kHz	4.8 kHz	100 Hz
100 kHz	19.2 kHz	2.4 kHz	40 Hz

The sampling sequence does not start immediately after the command is issued, but after the selected transition (L to H or H to L) is detected. While waiting for the transition to occur, the RB2 output is continuously held in the state which is opposite of the triggering logic level. This enables application on the wired-or logic, even if it is without pull-ups. If this condition never occurs, it is possible to escape by pressing the right key. In this case, message "Break" is displayed in the LCD upper row.

LED P has an additional function while sampling in analyzer mode. It is turned OFF when the start command is issued, then turned ON when sampling or the receiving condition was met, and then OFF again when all samples are fetched. In slower rates, it is noticeable that LED P blinks while sampling. One blinking period is equal to 32 sampling periods.

FIGURE 7: LOGIC ANALYZER SCHEMATIC

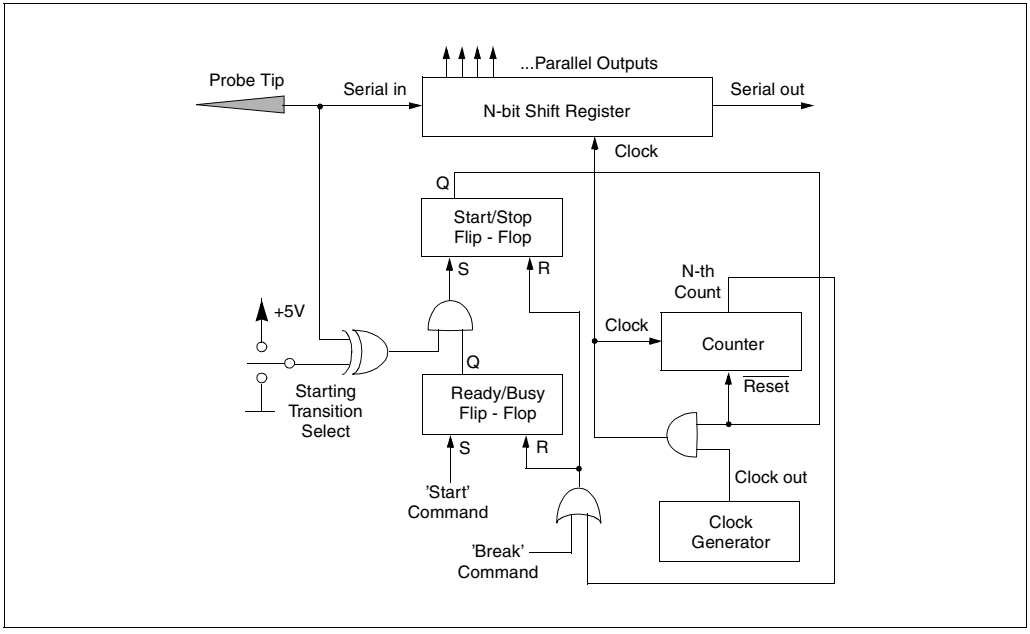
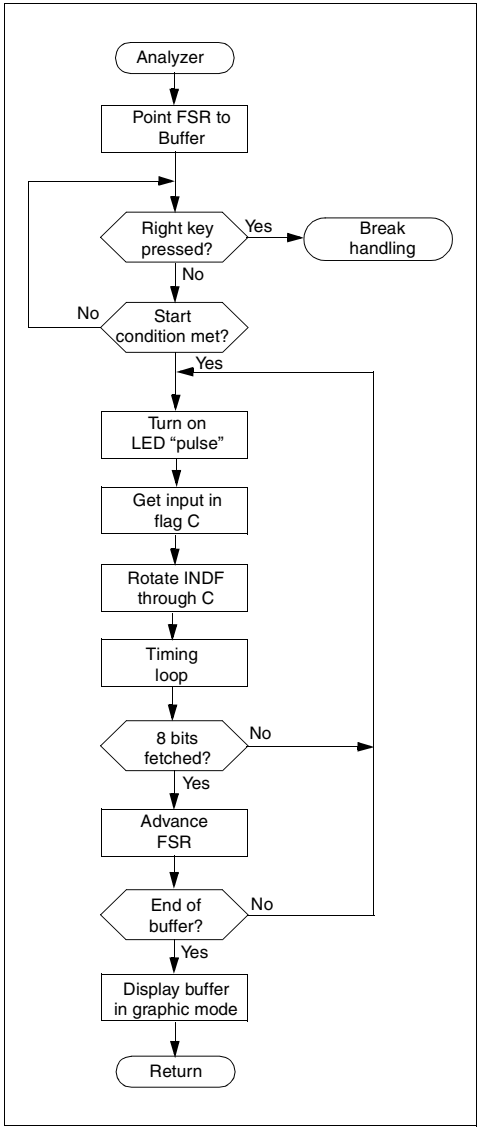


FIGURE 8: LOGIC ANALYZER FLOWCHART



All sample rates are generated by software, and the three highest ones use individual subroutines. The sample rate for 1 MHz, which is at the very beginning of the program, has to fetch and memorize a single bit sample by rotating it into the buffer, change the destination address after every 8 samples and exit the loop after 304 samples. All this while keeping uniform timing of 2 and 3 (alternated, which gives an average of 2.5) instruction cycles for one fetch. That could not be realized in a conventional manner, so it has a location-sensitive structure. Upon exit, it jumps to address 4Dh (which is far from the subroutine itself). If you modify anything in this program, take care not to affect this location.

The analyzer may have some unpredictable delays between an external starting event (rising or falling edge) and the first sample. In all cases, this delay may vary from 0 to 4 microseconds, so it may have some significance only in highest sample rates. One of the reasons for this delay is the time which the microcontroller requires for a key test, which enables the manual break if this event never comes. Also, there is some minor jitter at the 1 MHz analyzer sample rate. In the worst case, it might be 300 ns.

Serial Code Receiver

In this mode, a total of 42 bytes is received and displayed in both HEX and ASCII. The acceptable format is:

1 Start Bit / 7 or 8 Data Bits / 0 or 1 Parity Bit / 1 or more Stop Bits.

It is possible to connect the probe tip directly to the RS-232 or RS-232C +/- 12V voltage levels, to RS-422 or RS-485, or to +5V logic.

The available baud rates are:

1200	(1.2)
2400	(2.4)
4800	(4.8)
9600	(9.6)
19200	(19.2)
38400	(38.4)
57600	(57.6)
115200	(115)

7 or 8 data bits may be selected to adjust for the desired data format.

Parity or no parity bit (suffix "p"). This affects only the proper timing for this bit during reception. It is neither tested for validity nor displayed.

Standard RS232C or inverse polarity. If prefix "i" is displayed, then inverse polarity is active (low start bit, inverted data and optional parity bits and high stop bit). This is useful if the serial message must be fetched before the RS-232C TX drivers and after the RX buffers (which are both inverters).

Received bytes are displayed both in HEX and ASCII in 6 groups of 7 bytes each. ASCII representation is with bit 7 cleared, and the non-printable characters (00h-1Fh) are represented as dots. All other codes are standard ASCII.

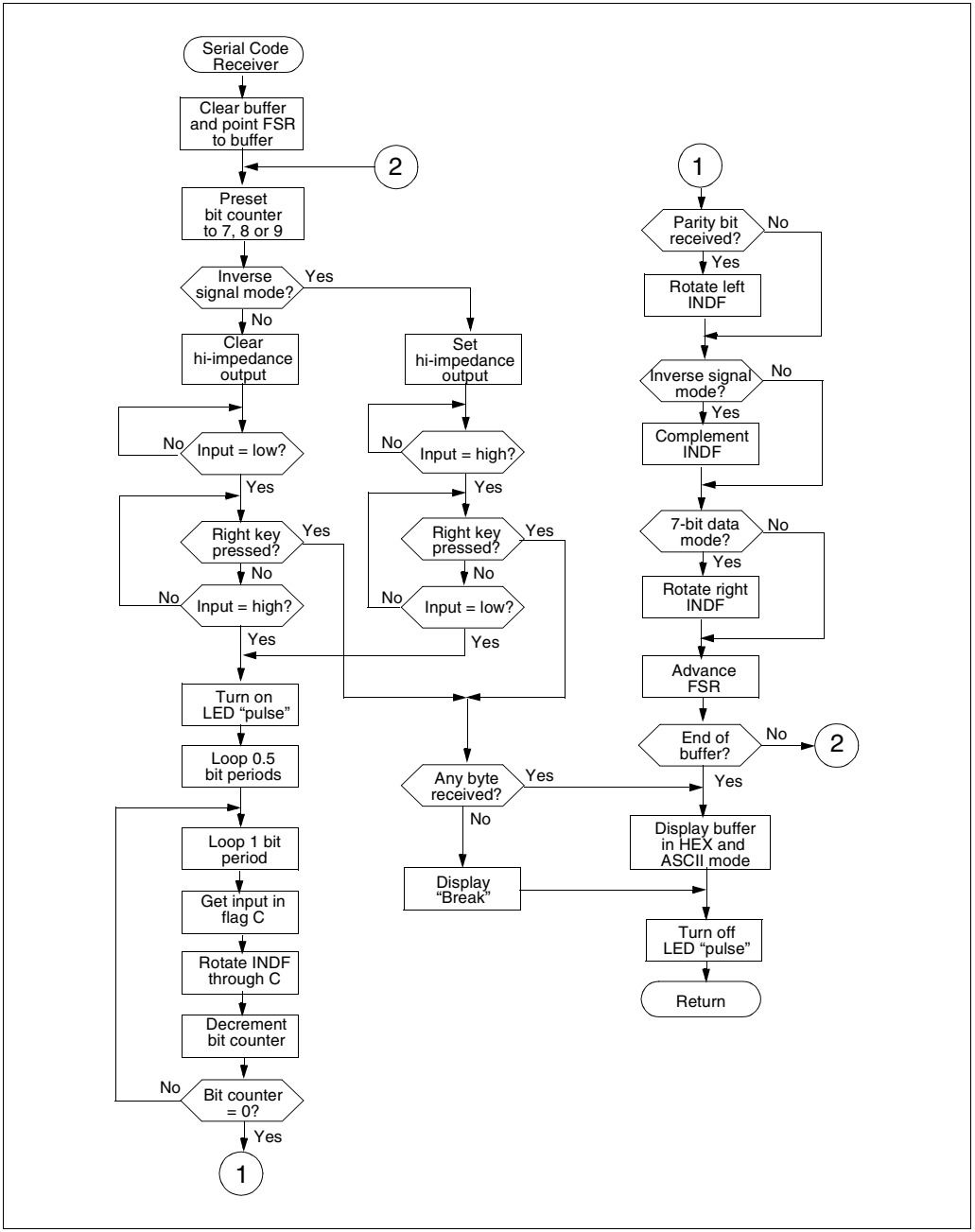
The string of received serial codes is 42 bytes long. If the string is shorter, the instrument will wait for next start bit, so it may look like it is stuck without any message. In that case, reception may be stopped by pressing the right key. If no bytes were received, the message "Break" will be displayed, but if at least one byte was received, the received sequence will be displayed with all unreceived bytes represented as zeros.

Similar to the analyzer mode, LED P will be turned ON when the first start bit is detected. This helps to detect sequences of less than 42 bytes in length.

No error test is performed during reception.

Figure 9 represents the flowchart for the serial code receiver.

FIGURE 9: SERIAL CODE RECEIVER FLOWCHART



Frequency Counter

Figure 10 shows the standard structure of the hardware solution for a frequency counter. All this is substituted by software in the PICmicro® microcontroller (MCU) aided by the existing TMR0. All counters are binary, and the counter state is displayed after a 4-byte binary to 8-digit decimal conversion. The display refresh rate is 2 Hz.

The flow chart for the frequency counter is represented in Figure 11. As this is the real-time function, the existing keyboard subroutine might not be used, but separate key and time-out tests are written. The logic probe function is disabled in this mode.

There is no "Start" command here, as this function is active all the time while the instrument is in Frequency Counter mode. There is only one submode, range select, so pressing the right button is not used for stepping through submodes, but it changes the range immediately.

Internal counter TMR0 is used, and the program expands the width of the counter for an additional two bytes. The fourth byte is added after 500 ms of counting and multiplying the 24-bit counter state by a constant, which depends on which prescaler factor was used.

The prescaler also affects the counter resolution. Here are the counter ranges and corresponding resolutions:

Range 5 MHz / Resolution 4
Range 10 MHz / Resolution 8
Range 20 MHz / Resolution 16
Range 40 MHz / Resolution 32

The resolution surely affects the reading error of the frequency counter, but this error is still less than the error which is caused by the initial non-accuracy of industrial class quartz crystals.

FIGURE 10: FREQUENCY COUNTER SCHEMATIC

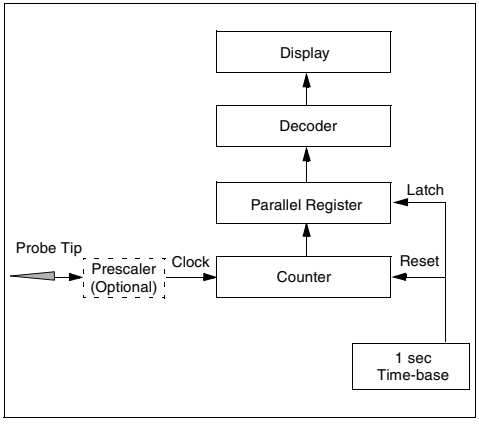
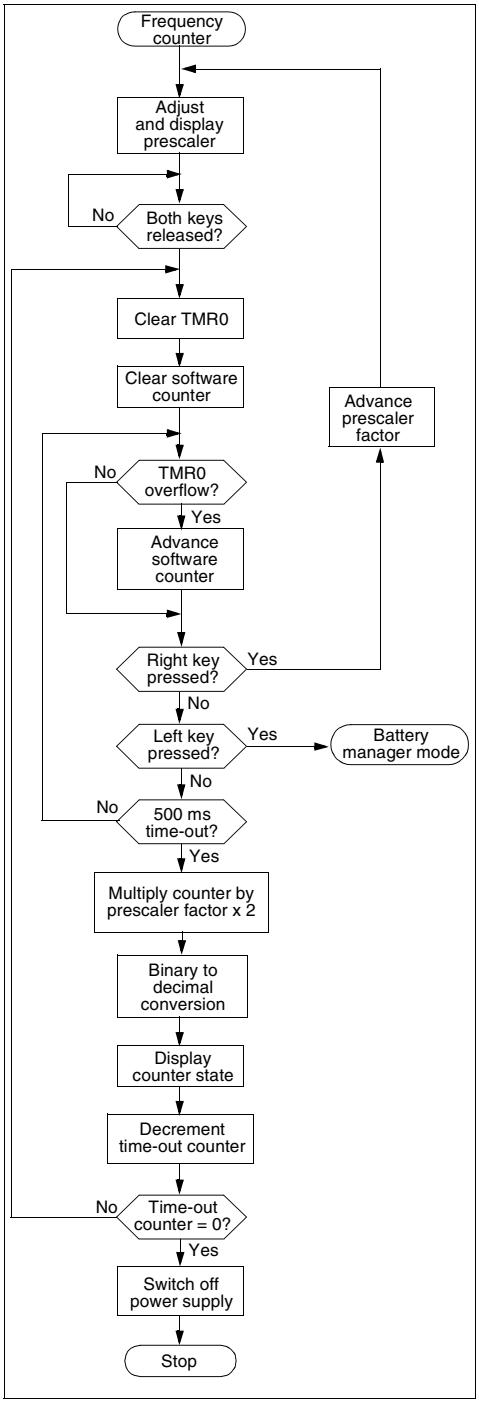


FIGURE 11: FREQUENCY COUNTER FLOWCHART



Battery Manager

The battery manager has three submodes. The first one is **manual power-off**, although there is also the automatic power off after approximately 8 minutes of inactivity (no key pressed).

The second submode is **discharging**. It is performed with 100 mA current through the resistors. The voltage monitor informs the PICmicro MCU if battery voltage is lower than 4V. If it is, the mode is automatically switched to charging. It is recommended that an external DC power supply be connected before the discharging command is issued. This will decrease the resulting discharging current to about 80 mA when the instrument is ON, and the DC supply is connected as the charging current flows independently of the mode selected.

Charge submode, when started, displays the time in HH:MM format, starting from 00:00, and switches off the instrument (and charging current also) at 14:00.

It is also possible to charge the NiCd battery even if it is not discharged, but this is not recommended, as unintentional overcharging may affect its capacity and life.

The unit is ready to charge the battery all the time if it is switched ON, even if the command charge is not active. It is enough to connect the external DC supply and to turn the instrument ON.

If the LCD were not counted, more than half of the hardware is used for discharging and charging. Figure 12 explains the structure of the battery manager hardware in a simplified form, where transistors T1, T4 and T5 are replaced by switches, for clarity. The flow chart for the battery discharger and charger is shown in Figure 13.

FIGURE 12: BATTERY MANAGEMENT SCHEMATIC

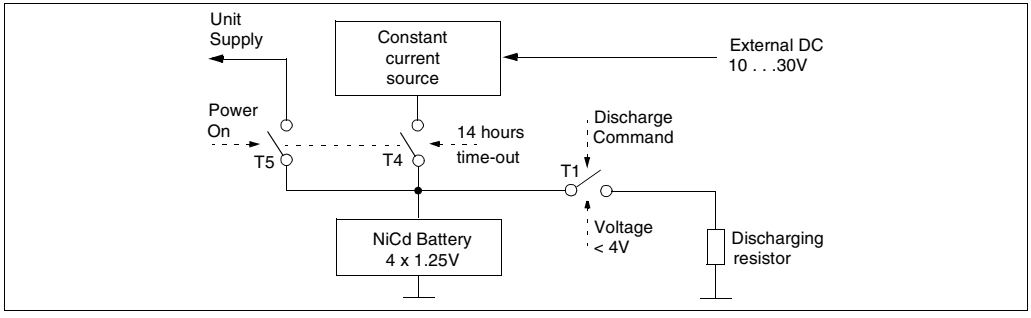
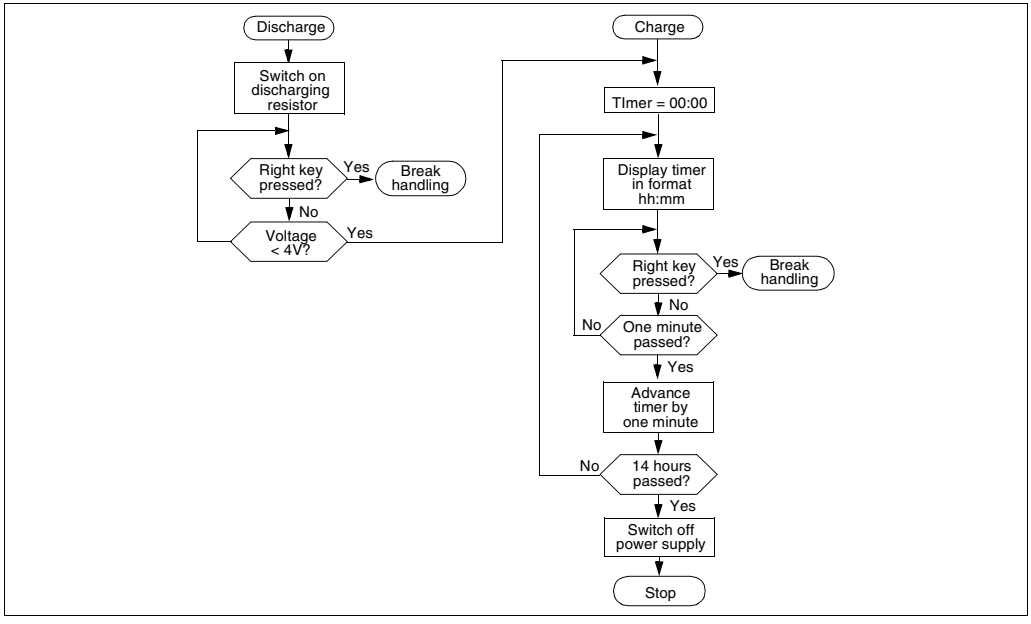


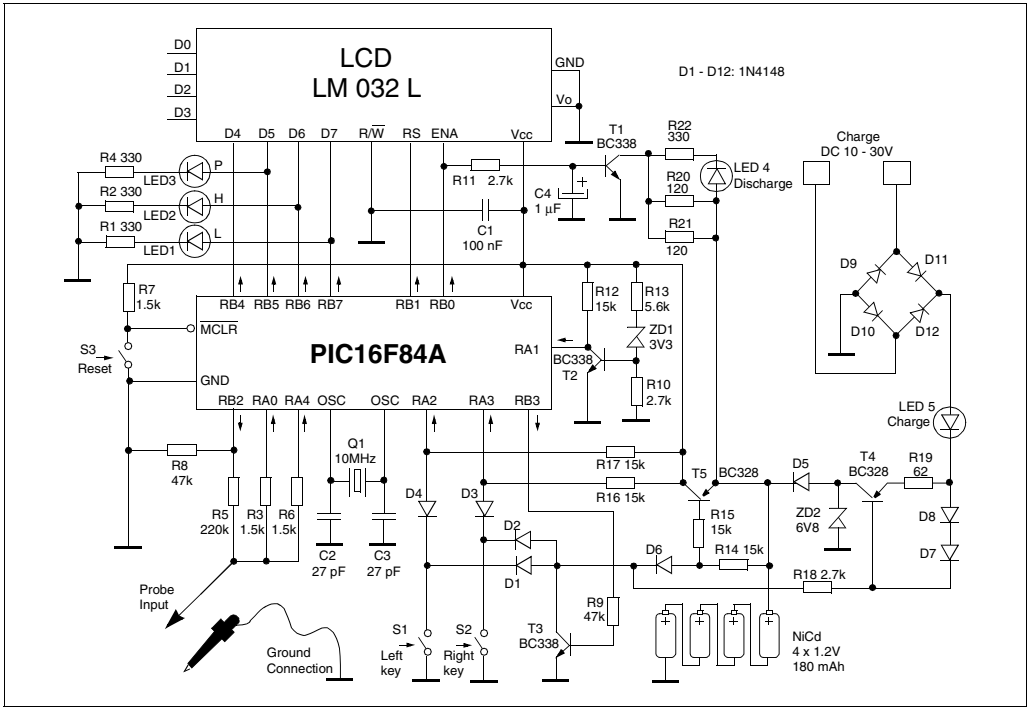
FIGURE 13: BATTERY MANAGER FLOWCHART



HARDWARE OVERVIEW

The complete schematic diagram is shown in Figure 14. The only IC used is the PIC16F84A-10/P, running at 10 MHz. It controls the intelligent LCD module via pins RB4-RB7, using two additional ports (RB0 and RB1) for the Enable signal and Register Select (Control/Data).

FIGURE 14: PIC16F84A PROBE SCHEMATIC



LCD module LM032L may be controlled in 4-bit parallel mode, which is used in this project. That is why pins 7-10 of the LCD are not used.

LEDs L, H and P share the same outputs with the LCD data bus. The only consequence is that some very minor and short LED flashing is visible while the PICmicro MCU is accessing the LCD (mostly when some key is pressed). **RB0** also has two functions. It controls the Enable signal for the LCD module and the transistor T1, which is used for the discharging function. To eliminate current spikes through T1 while accessing the LCD, capacitor C4 is added, which disables activating T1 by short Enable pulses. The main discharging resistors (R20 and R21) are intentionally located at two distant places on the PCB to minimize heat dissipation in a small area.

Inputs RA0 and RA4 are used to read the same signal, which was necessary because RA4 is the only port which may be directed to the `TMR0` counter, and RA0 had to be the "edge bit" in the port. The analyzer fetch routine in the highest rate must rotate `PORTA` bits in a single instruction and perform bit transfers from the input pin to the `STATUS,C` flag.

Resistors R3 and R6 are used for signal voltage limitation, with internal reverse diodes in the PIC16F84A. This enables connecting the probe tip directly to the RS-232C connector, which has +/- 12V voltage range. Resistor R8 disables input floating while the unit is OFF.

The voltage monitor (transistor T2 with ZD1) acts as a comparator which holds input port RA1 low if the battery voltage is higher than 4V. This is used to automatically detect the end of the discharging process. The value of resistor R13 can be modified to fine trim this cut-off voltage to 4V.

Transistor T5 is the main ON-OFF switch. **RB3** controls this transistor through T3. It is also activated by pressing the left or right key. Diodes D3 and D4 disable false activation of T5 through internal diodes in the PICmicro MCU when it is powered OFF.

Transistor T4 is the constant current regulator, which enables the use of any DC supply between 10V and 30V for battery charging. **Diodes D9-D12** allow application of any voltage polarity. **Zener diode ZD2** is not the voltage stabilizer, but it protects the hardware from overvoltage if the battery contacts are not properly tied while charging.

S3 is the RESET key, which is mounted on the solder side of the PCB, and is accessible from the lower side of the instrument through the small hole at the bottom plane of the package. It is used if the MCU drops into a deadlock state for some reason or when the unit is switched ON for the first time after assembly.

Pin 3 (Vo) on the LCD connector is for LCD driving voltage. The manufacturer recommends the use of a potentiometer (10-20k) for voltage adjustment on this input and to fine trim the LCD contrast, but in all cases the contrast was optimal when the potentiometer was in its lower-most position (Vo shortened to GND). So it was rejected and pin 3 was connected to GND in the final version of this instrument.

Charging current will flow all the time while DC supply is connected and the unit is ON, even if the unit is not in Charge mode. When the unit is OFF (e.g. when the charging 14-hour process is finished), the charging current is stopped.

Both charging and discharging are indicated on individual LEDs.

Note:	The LCD module used is Hitachi's LM032L. Type LM032LT may also be used, but it is not recommended, as it is a transflective type and contains the integrated illumination (which may not be used in this case, as it requires high voltage). Do not use modules LM032H or LM032HT as they require a dual voltage supply (+5/-5V).

FIRST SWITCHING ON AFTER ASSEMBLING

The batteries should be connected last, as there is no easy way to disconnect them once they are soldered, also it is not recommended to assemble the hardware while the voltage is present. The best way is to test the instrument with some external 5V power supply, and when it is completely debugged and tested, the batteries may be soldered. Do not connect the external DC supply for charging batteries if the batteries are not safely in their places! Zener diode ZD2 will reduce the voltage to 6.8V, but avoid testing the efficiency of this protection if at all avoidable.

If the NiCd batteries are discharged to the point the PICmicro cannot operate, it will be necessary to keep the left or right key pressed for some time (while the DC supply is connected for charging), as the pressing of any key makes hardware bypass for charging current. After about one minute of such charging, the battery voltage will be sufficient and then the unit will probably need to be reset by pressing S3. The normal charging process should then be used by executing the Charge command in Battery mode.

The contrast on the LCD is voltage-dependent, and as there is no voltage stabilizer, it appears to be a little darker immediately after a full charge. The battery voltage will be slightly over 5V. This will not affect readability. After a few minutes of operation, the battery voltage stabilizes and the LCD appears as normal.

Note: Data EEPROM is used for some lookup tables. This is read-only data and the Data EEPROM must be programmed before the unit is ready to use. The MCU will not affect data EEPROM contents. If your programmer does not support automatic loading of Data EEPROM contents from the HEX file, it must be loaded manually (a total of 61 bytes are used, and the last three bytes are don't care). The following will help in that case (all values are hexadecimal):

```
addr 00-07h:   88 01 01 98 F4 02 8C E4
addr 08-0Fh:   2C 88 64 0A 88 32 14 D8
addr 10-17h:   80 1A 88 19 28 C8 C0 34
addr 18-1Fh:   88 0A 64 C8 60 68 C8 30
addr 20-27h:   D0 C9 18 A1 80 01 01 14
addr 28-2Fh:   90 19 00 64 0A 00 28 19
addr 30-37h:   01 06 09 10 16 2E 30 64
addr 38-3Ch:   CC 05 0E 3B 95
```

MECHANICAL CONSTRUCTION

The components layout is shown in Figure 16. All components are placed on the component side of the PCB, except key S3 (reset), which is on the solder side. So are the NiCd batteries, which are placed in the specially shaped PCB edges and soldered directly to the PCB.

The LCD module is placed on M3 spacers, 7 mm long, which are tightened to 5 mm long spacers at the bottom side of the PCB. This leaves enough room for batteries which are 10 mm in diameter. Key 3 should not be higher than 5 mm, and the recommended height for keys 1 and 2 is about 16 mm. As the keys listed in the parts list are 14.5 mm high, they should be mounted on an extra spacer about 1.5 mm thick, non-conductive material.

The probe tip is fixed using three wire loops soldered to the PCB and to the tip. If it is not possible to get a connector for the DC supply which fits to the PCB pads, it is also possible to cut the PCB (across the dotted line on the components layout) to make enough space for some other type of connector, which may be tightened to the package. Pads for wires, needed in this case, are provided on the PCB. The polarity is not significant.

It is possible to build the package of the same material which is used for printed circuit boards, as it can easily be cut and joints soldered. Figure 15 shows the package detail.

FIGURE 15: CASE/PCB CONSTRUCTION

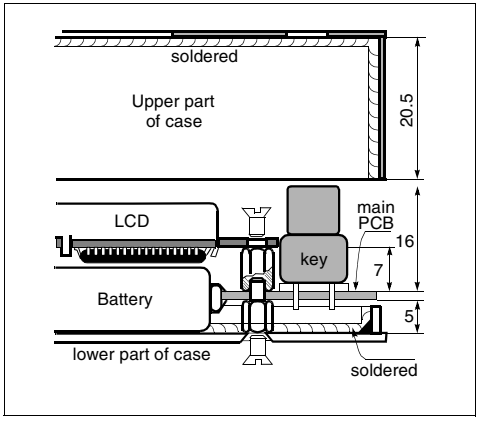
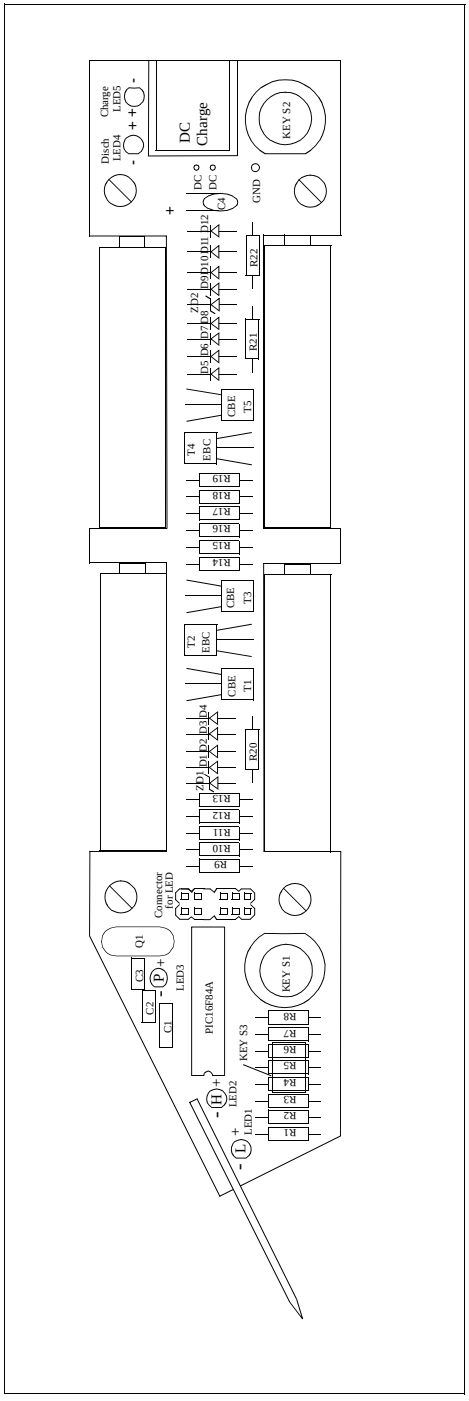


FIGURE 16: PCB PARTS LAYOUT



PARTS LIST

1. LCD module type LM032L (Hitachi) 1
2. PCB 1
3. Microcontroller PIC16F84A-10/P (Microchip) 1
4. Transistors:

   BC338 (or any small signal silicon NPN in SOT-54, pinning CBE)3

   BC328 (or any small signal silicon PNP in SOT-54, pinning CBE)2
5. Diodes:

   1N4148 (or any small signal silicon diode)12

   ZPD 6V8 (or any low power 6.8V zener diode)1

   ZPD 3V3 (or any low power 3.3V zener diode)1
6. Resistors:

   62R 1/4W axial1

   120R 1/4W axial 2

   330R 1/4W axial 4

   1K5 1/4W axial3

   2K7 1/4W axial3

   5K6 1/4W axial1

   15K 1/4W axial5

   47K 1/4W axial2

   220K 1/4W axial1
7. Capacitors:

   27 pF ceramic2

   100 nF ceramic1

   1 uF tantal1
8. Quartz:

   10 MHz 1
9. LEDs:

   red, 3 mm diameter 2

   green, 3 mm diameter2

   yellow, 3 mm diameter1
10. I.C. socket:

    18-pin 1
11. Keys:

    typ ITT D 6 (raster 5*5 mm, 14.5 mm high) 2

    typ SEL ET 5 (raster 5.5*3) or SEL ET 11 (raster 7.5*5)1
12. Connectors:

    2*7 pins male connector for PCB, raster 2.54 mm (100 mils)1

    2*7 pins female connector for PCB, raster 2.54 mm (100 mils)1

    cable-end crocodil-grip for GND connection1

    coaxial 3.5 mm female connector typ PG2031
13. Mechanical parts:

    spacer M3, 7 mm high 4

    spacer M3, 5 mm high 4

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;*******************************************************************
                     00002 ;* Filename: PROBE.ASM
                     00003 ;*******************************************************************
                     00004 ;* Author:   Voja Antonic
                     00005 ;* Company:  PC Press
                     00006 ;* Revision: RevA1
                     00007 ;* Date:     07-09-98
                     00008 ;* Assembled using MPASM rev 01.50
                     00009 ;*******************************************************************
                     00010 ;* Include files:
                     00011 ;*      p16f84.inc
                     00012 ;*******************************************************************
                     00013 ;* This is the program for multi-purpose laboratory instrument which
                     00014 ;*  consists of logic probe, single-channel logic state analyzer,
                     00015 ;*  serial code receiver and frequency counter. As this is single-chip
                     00016 ;*  instrument, all functions are supported by software.
                     00017 ;* LCD module used is Hitachi's LM032L with 2 lines of 20 columns.
                     00018 ;*
                     00019 ;* Note: The code is optimized for code space, and for that reason the
                     00020 ;*  most of code could not be written in modular format. For the same
                     00021 ;*  reason a lot of subroutines have more than one entry point and some
                     00022 ;*  of them are terminated by GOTO instead of RETURN.
                     00023 ;*
                     00024 ;* I/O port usage (all PORTA bits are inputs, all PORTB bits outputs)
                     00025 ;*  (note: bits 0, 5, 6 and 7 in port B have two functions each):
                     00026 ;*  PORTA,0 (input)   probe input
                     00027 ;*  PORTA,1 (input)   voltage monitor (high if battery voltage < 4 V)
                     00028 ;*  PORTA,2 (input)   left key (key 1) (low = key pressed)
                     00029 ;*  PORTA,3 (input)   right key (key 2) (low = key pressed)
                     00030 ;*  PORTA,4 (input)   probe input
                     00031 ;*  PORTB,0 (output)  enable LCD (high=select LCD),discharge (high=on)
                     00032 ;*  PORTB,1 (output)  register select LCD (low=instruction,high=data)
                     00033 ;*  PORTB,2 (output)  hi-imp output for probe pin
                     00034 ;*  PORTB,3 (output)  current hold for MCU supply (low = off)
                     00035 ;*  PORTB,4 (output)  LCD module D4
                     00036 ;*  PORTB,5 (output)  LCD module D5, LED P (high = on)
                     00037 ;*  PORTB,6 (output)  LCD module D6, LED H (high = on)
                     00038 ;*  PORTB,7 (output)  LCD module D7, LED L (high = on)
                     00039 ;*
                     00040 ;* External Clock Frequency: 10 MHz
                     00041 ;* Config Bit Settings: CP=OFF, PWRTE=ON, WDT=OFF, OSC=HS
                     00042 ;* Program Memory Usage: 1023 words
                     00043 ;* Data RAM Usage: 68 bytes
                     00044 ;* Data EEPROM Usage: 61 bytes
                     00045 ;*  Note: This is read-only data, so the Data EEPROM must be programmed
                     00046 ;*  before the unit is used. MCU will not affect data EEPROM contents.
                     00047 ;*******************************************************************
                     00048         list p=16f84, f=inhx8m, n=0
                     00049         include "p16f84.inc"
                     00001         LIST
                     00002 ; P16F84.INC Standard Header File, V 2.00 Microchip Technology, Inc.
                     00136         LIST
                     00050
0000000C             00051 FLAG    equ     0ch     ; 1 by  flag register
0000000D             00052 RXBITS  equ     0dh     ; 1 by  bit0=parity,bit1=7/8 bits,bit 2=inverse
0000000E             00053 DJNZ    equ     0eh     ; 1 by  general purpose, e.g. loop counter
0000000F             00054 SCRATCH equ     0fh     ; 1 by  general purpose scratchpad
00000010             00055 PCOUNT  equ     10h     ; 1 by  timing count for led P (monostable sim)
00000011             00056 SUBMODE equ     11h     ; 1 by  submode (cursor horizontal position)
00000012             00057 DEBO1   equ     12h     ; 1 by  rotor for key 1 debouncing
00000013             00058 DEBO2   equ     13h     ; 1 by  rotor for key 2 debouncing
```

```
  00000014      00059 COUNT   equ     14h     ; 1 by  general purpose counter
  00000015      00060 RATE    equ     15h     ; 1 by  analyzer sample rate, 0...15
  00000016      00061 CHARCOU equ     16h     ; 1 by  char counter for fixed format display
  00000017      00062 SHOWCOU equ     17h     ; 1 by  1-4, which group of 60 samples is shown
  00000018      00063 DELAYL  equ     18h     ; 1 by  delay for led P on when led L is on
  00000019      00064 DELAYH  equ     19h     ; 1 by  delay for led P on when led H is on
  0000001A      00065 PRESC   equ     1ah     ; 1 by  prescaler rate for frequency counter
  0000001B      00066 TIMOUTL equ     1bh     ; 1 by  timeout counter lo, for auto power off
  0000001C      00067 TIMOUTH equ     1ch     ; 1 by  timeout counter hi, for auto power off
  0000001D      00068 RXRATE  equ     1dh     ; 1 by  rx baud rate, 0...7
  0000001E      00069 BIN4    equ     1eh     ; 4 by  arith buf bin value, lo byte first
  00000022      00070 CMP4    equ     22h     ; 4 by  arith buf for comparing, lo byte first
  00000026      00071 BUFFER  equ     26h     ;42 by  42 by receive buf for analyzer and RX
  00000001      00072 REL     equ     1       ;=1 to put cursor on 1st char of command
                00073                         ;=0 to put cursor before the command
                00074
                00075 ; Bits definitions for FLAG register (bit 0 not used):
  00000001      00076 DP      equ     1       ; decimal point in 3-digit bin2dec conv
  00000002      00077 PTIP    equ     2       ; prev.state of probe input (for edge detect)
  00000003      00078 RIPPLE  equ     3       ; zero blanking bit
  00000004      00079 XTOX    equ     4       ; analyzer start at: 1=rising, 0=falling edge
  00000005      00080 LEDP    equ     5       ; led Pulse, 1=on
  00000006      00081 LEDH    equ     6       ; led High, 1=on
  00000007      00082 LEDL    equ     7       ; led Low, 1=on
                00083
                00084 ;*******************************************************************
                00085 ;* Reset vector
                00086 ;*******************************************************************
0000 28A8       00087         goto    Start
                00088
                00089 ;*******************************************************************
                00090 ;* Get1MHz
                00091 ;* This subroutine fetches 307 samples (last 7 will be ignored) from
                00092 ;*  PORTA.0 rotating through CARRY at 1 MHz rate - 2.5 instr. cycles
                00093 ;*  for each sample, realized mostly as 2 and 3 cycles alternatively,
                00094 ;*  at the following order:
                00095
                00096 ;*  4t-2t-2t (not in main loop, executed only once), and then
                00097 ;*  2t-2t-3t-2t-3t-2t-3t-2t-3t-2t-3t-2t-3t-2t-2t-4t  (repeat 19 times)
                00098 ;*
                00099 ;* Call Common inits loop counter (COUNT) to make 19 cycles before
                00100 ;*  exiting (16 samples are fetched at each pass), and FSR to point to
                00101 ;*  BUFF. It also presets T0SE bit depending on XTOX bit (in FLAG reg)
                00102 ;*  to enable proper edge detect, as it will affect TMR0 state.
                00103 ;*  State of key 2 (Break) is tested while waiting for start condition.
                00104 ;* Write ptr FSR is incremented after every 8 samples. COUNT initial
                00105 ;*  value is 01101101, after ANDing 0c0h and subtracting 33h from it,
                00106 ;*  makes 0dh, even if COUNT is incremented 18 times. After 19 passes,
                00107 ;*  COUNT is incremented to b'10000000', which after AND 0c0h and
                00108 ;*  SUB 33h makes 4dh. Those jumps are location sensitive, and it makes
                00109 ;*  the whole subroutine unrelocateable.
                00110 ;* Between this subroutine and the instruction goto Finished (below),
                00111 ;*  which must be at loc. 4dh, there are 25 free locations. They are
                00112 ;*  used for tables DecTab and CurTab, which causes that those tables
                00113 ;*  must have the fixed length. If anything relocates here, take care
                00114 ;*  not to affect location of instruction goto Finished.
                00115 ;* Input/Output variables: None
                00116 ;*******************************************************************
0001            00117         org     1               ; this subroutine must start at addr 1
                00118
0001            00119 Get1MHz                         ; 2.5 t read cycle
0001 306D       00120         movlw   80h-.19         ; loop end in 19 cyc(38 by=304 smpls)
0002 226F       00121         call    Common          ; initialize COUNT, FSR, hi-imp out...
                00122                                 ; ...bit XTOX and T0SE bit
0003            00123 GetEdge
0003 1D85       00124         btfss   PORTA,3         ; test status of key 2 and...
```

```
0004 28CC      00125         goto    Break           ; ...jump to Break routine if pressed
               00126
0005 0801      00127         movf    TMR0,W          ; TMR0 = logic level edge detector
0006 0C85      00128         rrf     PORTA,F  ; <--- ; the first sample is a little earlier
               00129                                 ; ... to compensate starting delay
0007 1903      00130         btfsc   STATUS,Z        ; test if there was egde...
0008 2803      00131         goto    GetEdge         ; ...and loop if not
               00132
0009 0C80      00133         rrf     INDF,F          ; rotate bit into destination byte
000A 0C85      00134         rrf     PORTA,F  ; <--- get bit from input to C
000B 0C80      00135         rrf     INDF,F          ; rotate bit into destination byte
000C 0C85      00136         rrf     PORTA,F  ; <--- get bit from input to C
               00137                                 ; movwf PCL will jump here at 18 passes
               00138                                 ; ...@addr 0100 0000 (40h) - 33h = 0dh
000D 0C80      00139         rrf     INDF,F          ; rotate bit into destination byte
000E 0C85      00140         rrf     PORTA,F  ; <--- get bit from input to C
000F 0C80      00141         rrf     INDF,F          ; rotate bit into destination byte
0010 0C85      00142         rrf     PORTA,F  ; <--- get bit from input to C
0011 0C80      00143         rrf     INDF,F          ; rotate bit into destination byte
0012 0C85      00144         rrf     PORTA,F  ; <--- get bit from input to C
0013 0C80      00145         rrf     INDF,F          ; rotate bit into destination byte
               00146
0014 0A94      00147         incf    COUNT,F         ; COUNT = loop counter
               00148
0015 0C85      00149         rrf     PORTA,F  ; <--- get bit from input to C
0016 0C80      00150         rrf     INDF,F          ; rotate bit into destination byte
0017 0C85      00151         rrf     PORTA,F  ; <--- get bit from input to C
0018 0C80      00152         rrf     INDF,F          ; rotate bit into destination byte
               00153
0019 0A84      00154         incf    FSR,F           ; must be exactly 8 read cycles apart
               00155                                 ; ... between FSR incrementing
001A 0C85      00156         rrf     PORTA,F  ; <--- get bit from input to C
001B 0C80      00157         rrf     INDF,F          ; rotate bit into destination byte
001C 0C85      00158         rrf     PORTA,F  ; <--- get bit from input to C
001D 0C80      00159         rrf     INDF,F          ; rotate bit into destination byte
               00160
001E 0814      00161         movf    COUNT,W         ; COUNT = loop counter
               00162
001F 0C85      00163         rrf     PORTA,F  ; <--- get bit from input to C
0020 0C80      00164         rrf     INDF,F          ; rotate bit into destination byte
0021 0C85      00165         rrf     PORTA,F  ; <--- get bit from input to C
0022 0C80      00166         rrf     INDF,F          ; rotate bit into destination byte
               00167
0023 39C0      00168         andlw   0c0h            ; this will make first 18 jumps to 0dh,
               00169                                 ; ...and the 19th one to 4dh
0024 0C85      00170         rrf     PORTA,F  ; <--- get bit from input to C
0025 0C80      00171         rrf     INDF,F          ; rotate bit into destination byte
0026 0C85      00172         rrf     PORTA,F  ; <--- get bit from input to C
0027 0C80      00173         rrf     INDF,F          ; rotate bit into destination byte
               00174
0028 3ECD      00175         addlw   -33h            ; this will make first 18 jumps to 0dh,
               00176                                 ; ...and the 19th one to 4dh
0029 0C85      00177         rrf     PORTA,F  ; <--- get bit from input to C
002A 0C80      00178         rrf     INDF,F          ; rotate bit into destination byte
002B 0C85      00179         rrf     PORTA,F  ; <--- get bit from input to C
002C 0C80      00180         rrf     INDF,F          ; rotate bit into destination byte
               00181
002D 0A84      00182         incf    FSR,F           ; must be exactly 8 read cycles apart
               00183                                 ; ... between FSR incrementing
002E 0C85      00184         rrf     PORTA,F  ; <--- get bit from input to C
002F 0C80      00185         rrf     INDF,F          ; rotate bit into destination byte
0030 0C85      00186         rrf     PORTA,F  ; <--- get bit from input to C
0031 0C80      00187         rrf     INDF,F          ; rotate bit into destination byte
0032 0C85      00188         rrf     PORTA,F  ; <--- get bit from input to C
               00189
0033 0082      00190         movwf   PCL      ; jumps to 0dh in first 18 passes
```

```
                    00191                           ; jumps to 4dh at 19th pass
                    00192
                    00193
;*********************************************************************
                    00194 ;* This table is used for bin2ascii (4-byte to 8-digit) conversion
                    00195
;*********************************************************************
0034                00196 DecTab
0034  3498           3496                  3480        00197         dt      098h,096h,080h  ;
decimal 10 000 000
0037  340F 3442 3440  00198         dt      00fh,042h,040h  ; decimal 1 000 000
003A  3401 3486 34A0  00199         dt      001h,086h,0a0h  ; decimal 100 000
003D  3400 3427 3410  00200         dt      000h,027h,010h  ; decimal 10 000
0040  3400 3403 34E8  00201         dt      000h,003h,0e8h  ; decimal 1 000
0043  3400 3400 3464  00202         dt      000h,000h,064h  ; decimal 100
0046  3400 3400 340A  00203         dt      000h,000h,00ah  ; decimal 10
                    00204
                    00205 ;*********************************************************************
                    00206 ;* Cursor position table for all SUBMODEs in mode 4 (battery manager)
                    00207 ;*********************************************************************
0049                00208 CurTab4
0049  34D3 34C0 34C4  00209         dt      0d2h+REL,0c0h,0c3h+REL,0cah+REL
      34CB
                    00210
                    00211 ;*********************************************************************
                    00212 ;* This is exit point for subroutine Get1MHz. Do not move this
                    00213 ;*  instruction, it must be at address 4dh!
                    00214 ;*********************************************************************
004D                00215         org     80h-33h         ; addr 1000 0000 (80h) - 33h = 4dh
                    00216                                 ; Get1MHz jumps here
004D  2A6A          00217         goto    Finished
                    00218
                    00219 ;*********************************************************************
                    00220 ;* Table for LCD module character generator redefinition, to enable
                    00221 ;*  pseudographic representation of bit samples in analyzer mode. It
                    00222 ;*  defines first 8 characters, which are stored in LCD module's RAM.
                    00223 ;*********************************************************************
004E                00224 Graphs
                    00225         ;       ...  ..*  .*.  .**  *..  *.*  **.  ***
004E  3400 3401 3404  00226         dt      00h, 01h, 04h, 05h, 10h, 11h, 14h, 15h
      3405 3410 3411
      3414 3415
                    00227
                    00228 ;*********************************************************************
                    00229 ;* Cursor position table for all SUBMODEs in mode 1 (Analyzer)
                    00230 ;*********************************************************************
0056                00231 CurTab1
0056  34D3 34C0 34CD0 00232         dt      0d2h+REL,0c0h,0cch+REL,0ceh+REL,0d0h+REL
      34CF 34D1
                    00233
                    00234 ;*********************************************************************
                    00235 ;* Cursor position table for all SUBMODEs in mode 2 (Serial rcvr)
                    00236 ;*********************************************************************
005B                00237 CurTab2
005B  34D3 34C8 34CD  00238         dt      0d2h+REL,0c7h+REL,0cch+REL,0ceh+REL,0d0h+REL
      34CF 34D1
                    00239
                    00240 ;*********************************************************************
                    00241 ;* Prescaler table for resolution display in mode 3 (freq counter)
                    00242 ;*********************************************************************
0060                00243 PrescTab
0060  3404 3408 3410  00244         dt      .4, .8, .16, .32
      3420
                    00245
                    00246 ;*********************************************************************
                    00247 ;* Range table for max. frequency display in mode 3 (freq counter)
```

```
                        00248 ;*******************************************************************
0064                    00249 RangeTab
0064  3405 340A 3414     00250         dt      .5, .10, .20, .40
      3428
                        00251 ;*******************************************************************
                        00252 ;* Timing constants for serial code receiver
                        00253 ;*******************************************************************
0068                    00254 BaudRate
0068  34BC 345E 342E     00255         dt      .188, .94, .46, .23, .11, .5, .3, .1
      3417 340B 3405
      3403 3401
                        00256
                        00257 ;*******************************************************************
                        00258 ;* Text strings (terminator = last character with bit 7 set)
                        00259 ;*******************************************************************
0070                    00260 TxtHz
0070  344D 3448 347A     00261         dt      "MHz",'/'+80h
      34AF
0074                    00262 DisTxt
0074  344F 3466 3466     00263         dt      "Off Disch. Charg",'e'+80h
      3420 3444 3469
      3473 3463 3468
      342E 3420 3443
      3468 3461 3472
      3467 34E5
0085                    00264 Head1
0085  3441 346E 3461     00265         dt      "Analyze",'r'+80h
      346C 3479 347A
      3465 34F2
008D                    00266 Head2
008D  3453 3465 3472     00267         dt      "Seria",'l'+80h
      3469 3461 34EC
0093                    00268 Head3
0093  3446 3472 3465     00269         dt      "Frequenc",'y'+80h
      3471 3475 3465
      346E 3463 34F9
009C                    00270 Head4
009C  3442 3461 3474     00271         dt      "Batter",'y'+80h
      3474 3465 3472
      34F9
00A3                    00272 BrkMes
00A3  3442 3472 3465     00273         dt      "Brea",'k'+80h
      3461 34EB
                        00274
                        00275 ;******* START
                        00276 ;*******************************************************************
                        00277 ;* Power Up sequence:I/O port B defined as all outputs,PORTB,3 set to
                        00278 ;*  switch power supply on and the internal Data Ram is cleared
                        00279 ;*******************************************************************
00A8                    00280 Start
00A8  1683              00281       bsf     STATUS,RP0
00A9  0186              00282       clrf    TRISB           ; portb: all bits outputs,port a:inputs
00AA  1283              00283       bcf     STATUS,RP0
00AB  300C              00284       movlw   0ch             ; start of RAM clr, & PORTB output byte
00AC  0086              00285       movwf   PORTB           ; switch power supply ON (set PORTB,3)
00AD  23B8              00286       call    ClrRam          ; clear internal RAM and wait 33.8 ms
                        00287
                        00288 ;*******************************************************************
                        00289 ;* LCD module initialization. 4-bit mode selected, display data RAM
                        00290 ;* cleared cursor set to blink mode, and the pseudographics character
                        00291 ;*  for character set 00h-07h preset from table Graphs.
                        00292 ;*******************************************************************
00AE  1086              00293         bcf     PORTB,1       ; rs lo (instruction)
00AF  3002              00294         movlw   2             ; 4-bit mode
00B0  23E9              00295         call    Nibble        ; write 4-bit mode command
00B1  3028              00296         movlw  28h          ; func set: 4 bit mode,2 lines,5*7 dots
```

```
00B2  23CF            00297         call    WrComL      ; write command and wait 130 us
00B3  3006            00298         movlw   06h         ; modeset: cursor moves right, no shift
00B4  23CF            00299         call    WrComL      ; write command and wait 130 us
00B5  300D            00300         movlw   0dh         ; disp on, no cursor,blink cursor pos
00B6  23CF            00301         call    WrComL      ; write command and wait 130 us
                      00302
00B7  3040            00303         movlw   40h         ; cg ram addr 0
00B8  23CF            00304         call    WrComL      ; write address and wait 130 us
00B9  304E            00305         movlw   Graphs      ; start addr of graph set for lcd disp
00BA  008F            00306         movwf   SCRATCH     ; move start address to pointer
00BB  3020            00307         movlw   .8*.4       ; 8 special characters to define
                      00308                             ; CHARCOU decremented in Char routine,
                      00309                             ; ...that is why here is 8*4
00BC  0096            00310         movwf   CHARCOU     ; loop counter for 8 special characters
00BD                  00311         GoGraph
00BD  3005            00312         movlw   .5          ; five rows are equal
00BE  0094            00313         movwf   COUNT       ; counter for 5 rows
00BF                  00314         FiveRows            ; inner loop: 5 rows are equal
00BF  23B5            00315         call    PclSub1     ; move SCRATCH to PCL
00C0  23D9            00316         call    CharNCC     ; rows 1-5 from the table
00C1  0B94            00317         decfsz  COUNT,F     ; five passes over?
00C2  28BF            00318         goto    FiveRows    ; no, loop
                      00319
00C3  3015            00320         movlw   15h         ; 15h=b'10101'=dot-space-dot-space-dot
00C4  23D6            00321         call    CharBl      ; row 6:all dots set,row 7:all dots clr
00C5  23D7            00322         call    Blank       ; row 8: all dots cleared
00C6  0A8F            00323         incf    SCRATCH,F   ; inc ptr
00C7  0B96            00324         decfsz  CHARCOU,F   ; 8 characters defined?
00C8  28BD            00325         goto    GoGraph     ; no, loop 8 x
                      00326
                      00327 ;*******  USER INTERFACE
                      00328 ;*****************************************************************
                      00329 ;* This is home point for mode 1(Analyzer): prints text"Analyzer" and
                      00330 ;* command line in line 2, w/cursor location set on variable SUBMODE.
                      00331 ;* Then the keyboard routine is called, where it waits for key to be
                      00332 ;* pressed.
                      00333 ;* Break entry point prints message Break in line 1 and redraws line 2
                      00334 ;*****************************************************************
00C9                  00335 Mode1                       ; Mode 1: Analyzer
00C9  3084            00336         movlw   Head1-1     ; start address of string -1
00CA  235F            00337         call    Headline    ; print "Analyzer"
00CB  28CD            00338         goto    Farm1       ; avoid "Break" message
00CC                  00339 Break                       ; Break entry pt (if Break in Mode 1)
00CC  23DB            00340         call    PrintBrk    ; print "Break"
00CD                  00341 Farm1
00CD  227D            00342         call    PrintM1     ; print string in line 2
00CE                  00343 Farm1B
00CE  3056            00344         movlw   CurTab1     ; get cursor table addr in analyze mode
00CF  2170            00345         call    CurPosKb    ; place cursor on proper position
                      00346                             ; test keys / probe input,service leds
                      00347                             ; return if key press (C:key1,NC:key2)
                      00348
                      00349 ;*****************************************************************
                      00350 ;* If key 1 pressed (C),SUBMODE is advanced (range 0...4,then wrapto 0
                      00351 ;* If key 2 pressed (NC), program vectors to corresponding routine
                      00352 ;*(except if SUBMODE=1,then the sample rate is advanced and displayed)
                      00353 ;*****************************************************************
00D0  1803            00354         btfsc   STATUS,C    ; test which key was pressed
00D1  28D4            00355         goto    Key1A       ; jump if key 1
                      00356                             ; continue if key 2 pressed
00D2  212A            00357         call    Range5      ; advance var SUBMODE in range 0...4
00D3  28CE            00358         goto    Farm1B      ; go wait next key
00D4                  00359 Key1A                       ; key 1 pressed
00D4  0B11            00360         decfsz  SUBMODE,W   ; test SUBMODE (set Z if SUBMODE=1)
00D5  28DF            00361         goto    NoRate      ; jump if SUBMODE <>1
                      00362                             ; continue if SUBMODE=1
```

```
00D6  0A95            00363         incf    RATE,F      ; advance sample rate
00D7  1215            00364         bcf     RATE,4      ; RATE range 0...15
                      00365
00D8  22B7            00366         call    ClrRow1     ; prepare line 1 to print sample rate #
00D9  0A15            00367         incf    RATE,W      ; readjust RATE from 0...15 to 1...16
00DA  2370            00368         call    Print255    ; print serial # of sample rate 1...16
00DB  28CD            00369         goto    Farm1       ; go redraw row 2, wait next command
                      00370
00DC                  00371 EdgeSet                     ; Change start cond(L-2-H or H-2-L)
00DC  3010            00372         movlw   10h         ; bit 4 is flag XTOX
00DD  068C            00373         xorwf   FLAG,F      ; change flag
00DE  28CD            00374         goto    Farm1       ; go redraw row 2, wait next command
00DF                  00375 NoRate
00DF  1811            00376         btfsc   SUBMODE,0   ; bit 0 will be set only if SUBMODE=3
00E0  2A0E            00377         goto    Mode1Go     ; if SUBMODE=3
00E1  1891            00378         btfsc   SUBMODE,1   ; bit 1 will be set only if SUBMODE=2
00E2  28DC            00379         goto    EdgeSet     ; if SUBMODE=2
00E3  1911            00380         btfsc   SUBMODE,2   ; bit 2 will be set only if SUBMODE=4
00E4  29D4            00381         goto    Mode1Show   ; if SUBMODE=4
                      00382                             ; if SUBMODE=0, program drops to Mode2.
                      00383
                      00384 ;*****************************************************************
                      00385 ;* This is home for mode 2 (RS232 receiver): prints text "Serial" and
                      00386 ;* command line in line 2,cursor placed depended on variable SUBMODE.
                      00387 ;* BrkRS entry point prints message Break in line 1 and redraws line 2
                      00388 ;* if 0 bytes are received, display first 7 bytes if any byte received
                      00389 ;*****************************************************************
00E5                  00390 Mode2                       ; mode 2: Serial receiver
00E5  308C            00391         movlw   Head2-1     ; start address of string -1
00E6  235F            00392         call    Headline    ; print "Serial"
00E7  28ED            00393         goto    Farm2       ; avoid "Break" message
00E8                  00394 BrkRS                       ; Break entry pt (if Break in mode 2)
00E8  0804            00395         movf    FSR,W       ; FSR points to write next rcvd byte
00E9  3A26            00396         xorlw   BUFFER      ; if FSR=literal BUFF the 0 bytes rcvd
00EA  1D03            00397         btfss   STATUS,Z    ; test if FSR = literal BUFFER
00EB  2AC6            00398         goto    Show2       ; no -some bytes received,show them
00EC  23DB            00399         call    PrintBrk    ; yes -no bytes received, print "Break"
                      00400
                      00401 ;*****************************************************************
                      00402 ;* Prints baud rate in KBaud on LCD
                      00403 ;*
                      00404 ;* Input variables: RXRATE in range 0...7
                      00405 ;* Output variables: CHARCOU decremented by num of characters printed
                      00406 ;*****************************************************************
00ED                  00407 Farm2
00ED  30C8            00408         movlw   0c8h        ; baud rate position on LCD
00EE  23CF            00409         call    WrComL      ; move cursor command
                      00410
00EF  108C            00411         bcf     FLAG,DP     ; no decimal point printing if RATE=0
00F0  019F            00412         clrf    BIN4+1      ; BIN4+1 is high byte for baudrate disp
                      00413
00F1  0A1D            00414         incf    RXRATE,W    ; move RXRATE from range 0...7 to 1...8
00F2  008E            00415         movwf   DJNZ        ; DJNZ = RXRATE+1
                      00416
00F3  3073            00417         movlw   .115        ; case RXRATE=7:then Baudrate=115 Kbaud
00F4  198E            00418         btfsc   DJNZ,3      ; test if DJNZ=8 (same as RXRATE=7)
00F5  2905            00419         goto    Lth256      ; yes, go case 115.2 (RXRATE=7)
                      00420
00F6  148C            00421         bsf     FLAG,DP     ; for rete 0...6 there is decimal point
00F7  149F            00422         bsf     BIN4+1,1    ; case RXRATE=6:is hi byte for 57.6
00F8  3040            00423         movlw   .576-.512   ; case RXRATE=6:is lo byte for 57.6
00F9  0A8E            00424         incf    DJNZ,F      ; DJNZ=RXRATE+2
00FA  198E            00425         btfsc   DJNZ,3      ; test if DJNZ=8 (same as RXRATE=6)
00FB  2905            00426         goto    Lth256      ; yes, go case 57.6 (RXRATE=6)
                      00427
00FC  019F            00428         clrf    BIN4+1      ; for rates 0...5 hi byte is zero
```

```
00FD  3003            00429         movlw  .3          ; constant for rates 0...5
00FE  009E            00430         movwf  BIN4        ; BIN4 will be rotated (mult by 2)
                      00431                            ; RXRATE+2 times to get 1.2 - 2.4 - 4.8
                      00432                            ; - 9.6 - 19.2 - 38.4
00FF                  00433 X2Loop
00FF  1003            00434         bcf    STATUS,C    ; clear bit C to get multiplying by 2
0100  0D9E            00435         rlf    BIN4,F      ; multiply low byte
0101  0D9F            00436         rlf    BIN4+1,F    ; multiply hibyte,that is 16-bit rotate
0102  0B8E            00437         decfsz DJNZ,F      ; test if RXRATE+2 times multiplied
0103  28FF            00438         goto   X2Loop      ; no, loop
0104  081E            00439         movf   BIN4,W      ; yes, get result to print it
0105                  00440 Lth256
0105  2372            00441         call   PrintBR     ; print baud rate,incl. decimal point
0106  23D7            00442         call   Blank       ; to delete last # from previous rate
                      00443
                      00444 ;*****************************************************************
                      00445 ;* Prints num bits to be received (7 or 8), with suffix "p" if parity
                      00446 ;* bit will be received (not written to RAM!), and with prefix "i" if
                      00447 ;* inverse input polarity is expected. Input variable RXBITS, bit0 set
                      00448 ;* if parity bit expected, bit 1 set if 8-bit word and bit 2 set if
                      00449 ;* inverse polarity (lo start bit,inverse data bits and high stop bit)
                      00450 ;*****************************************************************
0107  30CC            00451         movlw  0cch        ; bit# pos (7/8/7p/8p/i7/i8/i7p/i8p) -1
0108  23CF            00452         call   WrComL      ; write command
                      00453
0109  3020            00454         movlw  ' '         ; space: true polarity
010A  190D            00455         btfsc  RXBITS,2    ; let it be space if RXBITS,2 cleared
010B  3069            00456         movlw  'i'         ; "i": inverse polarity
010C  23D8            00457         call   Char        ; print blank or "i"
                      00458
010D  3037            00459         movlw  '7'         ; "7": 7 bits
010E  1C8D            00460         btfss  RXBITS,1    ; let it be 7 if RXBITS,1 set
010F  3038            00461         movlw  '8'         ; "8": 8 bits
0110  23D8            00462         call   Char        ; print "7" or "8" (bits)
                      00463
0111  3020            00464         movlw  ' '         ; space: no parity
0112  180D            00465         btfsc  RXBITS,0    ; let it be space if RXBITS,0 cleared
0113  3070            00466         movlw  'p'         ; "p": parity bit exists
0114  23D8            00467         call   Char        ; print "p" (parity bit) or blank
                      00468
                      00469 ;*****************************************************************
                      00470 ;* This call prints number of group displayed and "*" (execution) symb
                      00471 ;*****************************************************************
0115  22A5            00472         call   KaoAna      ; print rest of line - is the same as
                      00473                            ; on mode 1 (analyzer)
                      00474
                      00475 ;*****************************************************************
                      00476 ;* Places cursor on proper position (input variable SUBMODE) and calls
                      00477 ;* keyboard subroutine, where it will wait for key to be pressed
                      00478 ;*****************************************************************
0116  305B            00479         movlw  CurTab2     ; table with cursor positions
0117  2170            00480         call   CurPosKb    ; place cursor on proper pos
                      00481                            ; test keys / probe input,service leds
                      00482                            ; return if key press (C:key1,NC:key2)
                      00483
                      00484 ;*****************************************************************
                      00485 ;* If key1 pressed (C), SUBMODE is advanced (range 0..4, then wrap to 0
                      00486 ;* If key 2 pressed (NC), program vectors to corresponding routine
                      00487 ;*  (except if SUBMODE=1, then the Baud rate is advanced and displayed)
                      00488 ;*****************************************************************
0118  1803            00489         btfsc  STATUS,C    ; test which key was pressed
0119  291C            00490         goto   Key1B       ; jump if C set, means key 1 pressed
                      00491                            ; key 2 pressed
011A  212A            00492         call   Range5      ; increment SUBMODE in range 0...4
011B  28ED            00493         goto   Farm2       ; go redraw row2, wait for next command
011C                  00494         Key1B              ; key 1 pressed
```

```
011C  1911            00495         btfsc  SUBMODE,2   ; bit 2 is set only if SUBMODE = 4
011D  2AC1            00496         goto   Mode2Show   ; jump if SUBMODE = 4
                      00497
011E  1C91            00498         btfss  SUBMODE,1   ; bit1 cleared only if SUBMODE = 0 or 1
011F  2925            00499         goto   Sub01       ; jump if SUBMODE = 0 or SUBMODE = 1
                      00500
0120  1811            00501         btfsc  SUBMODE,0   ; bit 0 is set here only if SUBMODE = 3
0121  2AEA            00502         goto   Mode2Go     ; jump if SUBMODE = 3
                      00503                            ; drops here if SUBMODE = 2
0122  0A8D            00504         incf   RXBITS,F    ; advance RXBITS (command)
0123  118D            00505         bcf    RXBITS,3    ; RXBITS cycle in range = 0...7
0124  28ED            00506         goto   Farm2       ; go redraw row2, wait for next command
0125                  00507 Sub01
0125  1C11            00508         btfss  SUBMODE,0   ; bit 0 is set here only if SUBMODE = 0
0126  2B28            00509         goto   FreqEp      ; if SUBMODE =0,goto frequency entry pt
                      00510
0127  0A9D            00511         incf   RXRATE,F    ; if SUBMODE = 1 then advance RXRATE
0128  119D            00512         bcf    RXRATE,3    ; RXRATE cycle in range 0...7
0129  28ED            00513         goto   Farm2       ; go redraw row2, wait for next command
                      00514
                      00515 ;*********************************************************************
                      00516 ;* This subroutine increments variable SUBMODE,and if the result is >4
                      00517 ;* it wraps to 0
                      00518 ;*********************************************************************
012A                  00519 Range5                     ; increment SUBMODE in range 0...4
012A  0A91            00520         incf   SUBMODE,F   ; advance SUBMODE
012B  1911            00521         btfsc  SUBMODE,2   ; if SUBMODE,2 cleared then no overflow
012C  1C11            00522         btfss  SUBMODE,0   ; if SUBMODE,0 cleared then no overflow
012D  0008            00523         return             ; no overflow: return
012E  0191            00524         clrf   SUBMODE     ; SUBMODE cycle in range 0...4
012F  0008            00525         return             ; SUBMODE wrapped to 0, return
                      00526
                      00527 ;*********************************************************************
                      00528 ;* Mode4 is home point for mode 4 (off/discharge/charge): prints text
                      00529 ;* "Battery" and command line in line2, with cursor placed depended on
                      00530 ;* variable SUBMODE. Then keyboard routine is called, where it will
                      00531 ;* wait for key to be pressed
                      00532 ;* Break4 entry point prints message Break in line1 and readraws line2
                      00533 ;* ExitDis is the entry point if key 2 is pressed during discharging
                      00534 ;*********************************************************************
0130                  00535 ExitDis                    ; exit disch entry point,if disch Break
0130  23BD            00536         call   DisEna30    ; turn off PORTB,0 discharge transistor
0131                  00537 Break4                     ; exit Chg entry point, if Charge Break
0131  23DB            00538         call   PrintBrk    ; print "Break"
0132  2934            00539         goto   Contm4      ; avoid headline printing
0133                  00540 Mode4                      ; mode 4: discharge/charge
0133  2360            00541         call   Headline2   ; print "Battery"
0134                  00542 Contm4
0134  23CE            00543         call   Row2        ; move cursor to line 2
0135  3073            00544         movlw  DisTxt-1    ; point to message -1
0136  23DD            00545         call   Write       ; print Off Disch Charge
0137                  00546 Farm4
0137  3049            00547         movlw  CurTab4     ; point to cursor table for mode 4
0138  2170            00548         call   CurPosKb    ; place cursor @ Off/Disch/Charge/-->
                      00549                            ; test keys / probe input,service leds
                      00550                            ; return if key press (C:key1,NC:key2)
                      00551
                      00552 ;*********************************************************************
                      00553 ;* If key1 pressed (C), SUBMODE is advanced (range 0..3,then wrap to 0
                      00554 ;* If key 2 pressed (NC), program jumps to corresponding routine
                      00555 ;*********************************************************************
0139  1803            00556         btfsc  STATUS,C    ; test which key was pressed
013A  293E            00557         goto   Key1D       ; C set: key 1 pressed
                      00558                            ; key 2 pressed
013B  0A91            00559         incf   SUBMODE,F   ; advance SUBMODE
013C  1111            00560         bcf    SUBMODE,2   ; SUBMODE cycle in range 0...3
```

```
013D  2937            00561         goto    Farm4       ; go redraw row2, wait for next command
013E                  00562 Key1D                       ; key 1 pressed
013E  1891            00563         btfsc   SUBMODE,1   ; bit 1 set only if Chg/Disch. submode
013F  2943            00564         goto    ChargDis    ; goto Charge or Discharge process
                      00565                             ; depended on bit 0 in variable SUBMODE
0140  1C11            00566         btfss   SUBMODE,0   ; SUBMODE,0 cleared here if SUBMODE=0
0141  28C9            00567         goto    Mode1       ; shortcut to mode 1 (Analyzer)
                      00568
0142  29BC            00569         goto    Suicide     ; manual power off
                      00570
                      00571 ;*******  CHARGE/DISCHARGE
                      00572 ;*****************************************************************
                      00573 ;* Charge/Disch entry point. If bit SUBMODE,0 set, then go to Charge
                      00574 ;*  SUBMODE,1 is set in this point. (SUBMODE=2 or 3)
                      00575 ;*****************************************************************
0143                  00576 ChargDis
0143  22B7            00577         call    ClrRow1     ; in both cases row 1 must be cleared
0144  1811            00578         btfsc   SUBMODE,0   ; test if SUBMODE,0 set, if so...
0145  2952            00579         goto    Charge      ; ...jump to Charge (SUBMODE=3)
                      00580                             ; ...else continue to disch (SUBMODE=2)
                      00581
                      00582 ;*****************************************************************
                      00583 ;* Discharge starts here (SUBMODE=2)
                      00584 ;* Cursor moved to line 1 under text "Disch."
                      00585 ;* Then command 02h (Home Cursor) issued to LCD controller, but this
                      00586 ;* is dummy command - sense is to freeze it after first nibble, and
                      00587 ;* thus to leave PORTB,0 (ENA) in high state as long as discharging
                      00588 ;* lasts. After the discharging termination (if volt monitor detects
                      00589 ;* <4V or key 2 pressed), the command for LCD controller will be
                      00590 ;* completed, switching discharging transistor off.
                      00591 ;* If discharging is broken by key, program returns to user interface
                      00592 ;* for mode 4, if terminated by voltage monitor, charging takes place
                      00593 ;*****************************************************************
0146  1086            00594         bcf     PORTB,1     ; pull LCD Reg Select low (=instr)
0147  0103            00595         clrw                ; high nibble of instruction 02h = 0h
0148  23F1            00596         call    Hinib_B     ; output W,4-7 to 4-bit LCD data bus
0149  23ED            00597         call    EnaLCD      ; generate En signal (1200us) for LCD
014A  3020            00598         movlw   20h         ; command 02h=home cursor(swap nibbles)
                      00599
014B  23F1            00600         call    Hinib_B     ; output W,4-7 to 4-bit LCD data bus
014C  1406            00601         bsf     PORTB,0     ; ENA activated (the command won't be
                      00602                             ; finished until Break or voltage < 4V)
014D                  00603 DisLoop
014D  1D85            00604         btfss   PORTA,3     ; test key 2 status...
014E  2930            00605         goto    ExitDis     ; if low,disching manually broke by key
                      00606
014F  1C85            00607         btfss   PORTA,1     ; test voltage monitor...
0150  294D            00608         goto    DisLoop     ; if still >=4V, loop
                      00609                             ; dischging terminated (voltage < 4V)
                      00610
0151  23BD            00611         call    DisEna30    ; switch off discahge transistor
                      00612
                      00613 ;*****************************************************************
                      00614 ;* Charging starts here (by command or after successful discharging)
                      00615 ;* Minute and hour counters are init'd and counting process starts.
                      00616 ;* Clock(in format HH:MM) is displayed in line 1 under text "Charge".
                      00617 ;* If charging broken by key, program returns to user interface for
                      00618 ;*  mode 4, if terminated by timeout (14 hours), the unit jumps to
                      00619 ;*  SUICIDE (switches off the unit forcing the output PORTB,3 low).
                      00620 ;*****************************************************************
0152                  00621 Charge                      ; Charge entry point
                      00622
0152  019B            00623         clrf    TIMOUTL     ; initialize minute counter 0...59
0153  019C            00624         clrf    TIMOUTH     ; initialize hour counter 0...13
0154                  00625 ChLoop
0154  308B            00626         movlw   8bh         ; position of digital clock on LCD
```

```
0155  23CF            00627         call    WrComL      ; cursor to digital clock pos
0156  081C            00628         movf    TIMOUTH,W   ; TIMOUTH=hours in binary format
0157  2370            00629         call    Print255    ; print hour in format HH
0158  303A            00630         movlw   ':'         ; ":" = separator
0159  21D1            00631         call    PrintTL     ; print ":" and minute in format MM
                      00632
015A  30E4            00633         movlw   .228        ; 228 x 263270.4 us = 60 sec
015B  008F            00634         movwf   SCRATCH     ; high byte loop counter for 1 min loop
015C                  00635 Min1
015C  23D2            00636         call    GoLoop      ; 1283t (513.2us) inclusive     ; 1283t
015D  23D2            00637         call    GoLoop      ; total 1026.4us                ; 1283t
                      00638
015E  1D85            00639         btfss   PORTA,3     ; 2t  test status of key 2...
015F  2931            00640         goto    Break4      ; -   ...if low,chg manually terminated
                      00641
0160  0B94            00642         decfsz  COUNT,F     ; 1t  low byte loop counter
0161  295C            00643         goto    Min1        ; 2t  inner pass 1028.4 us
                      00644
0162  0B8F            00645         decfsz  SCRATCH,F   ; high byte loop counter for 1 minute
0163  295C            00646         goto    Min1        ; one pass 263270.4 us
                      00647
0164  0A9B            00648         incf    TIMOUTL,F   ; advance minute counter
0165  081B            00649         movf    TIMOUTL,W   ; TIMOUTL = minute up counter
0166  3EC4            00650         addlw   -.60        ; test if 60 minutes of charging done..
0167  1C03            00651         btfss   STATUS,C    ; if 60 minutes passed, C should be set
0168  296B            00652         goto    NotHour     ; not yet hour advance
0169  019B            00653         clrf    TIMOUTL     ; if 60 minutes done, clr minute cntr
016A  0A9C            00654         incf    TIMOUTH,F   ; ..and advance TIMOUTH=hour up counter
016B                  00655 NotHour
016B  081C            00656         movf    TIMOUTH,W   ; TIMOUTH = hour up counter
016C  3EF2            00657         addlw   -.14        ; test if 14 hours of charging
016D  1C03            00658         btfss   STATUS,C    ; if 14 hours passed, C should be set
016E  2954            00659         goto    ChLoop      ; ...if not yet 14 hours, loop
016F  29BC            00660         goto    Suicide     ; charging terminated after 14 h
                      00661
                      00662 ;*******  KEYBOARD AND PROBE
                      00663 ;*****************************************************************
                      00664 ;* CurPosKb
                      00665 ;* This subroutine places cursor in line 2 at position taken from the
                      00666 ;* lookup table: table offset is addressed by W at input, and table
                      00667 ;* read location by variable SUBMODE.
                      00668 ;* High timeout counter (TIMOUTH) is initialized. This sets automatic
                      00669 ;* Power Off timing to about 8 minutes. TIMOUTL is of minor importance
                      00670 ;* here (it affects less than 2 secs of timing), so it was not worth
                      00671 ;* waisting one instruction word.
                      00672 ;* Bit DEBO,0 set to disable false recognizing of PORTA,3 low level as
                      00673 ;* falling edge (as if key 2 was just pressed). This could happen if
                      00674 ;* some function was broken by pressing key2, as those are simple port
                      00675 ;* tests without affecting debouncer.
                      00676 ;* This subroutine continues to keyboard scan subroutine.
                      00677 ;*
                      00678 ;* Input variables: SUBMODE, affects cursor position
                      00679 ;* Output variables: TIMOUTH=0, high timeout counter
                      00680 ;*****************************************************************
0170                  00681 CurPosKb
0170  0711            00682         addwf   SUBMODE,W   ; add SUBMODE to lookup table offset
0171  23B6            00683         call    PclSub      ; get cursor position from table
0172  23CF            00684         call    WrComL      ; write new cursor position to LCD
0173  019C            00685         clrf    TIMOUTH
                      00686
0174  1412            00687         bsf     DEBO1,0     ; set any bit in both debouncers...
0175  1413            00688         bsf     DEBO2,0     ; ...to disable false recognizing of...
                      00689                             ; ...low level as falling edge
                      00690
                      00691 ;*****************************************************************
                      00692 ;* GoKbd
```

```
                     00693 ;* Is main keyboard subroutine, in which program loops all the time
                     00694 ;* except in freq counter mode, or while wait for start condition or
                     00695 ;* executing some command. Program exits subrout only if some key is
                     00696 ;* just pressed (not if continuosly pressed), or when timeout counter
                     00697 ;* (TIMOUTH, TIMOUTL) after 8 min reaches zero. If key 1 was pressed,
                     00698 ;* flag STATUS,C will be reset at exit, if key 2 was pressed, flag
                     00699 ;* STATUS,C will be set. If timeout detected, the routine SUICIDE is
                     00700 ;* executed (the unit is switched off forcing the output PORTB,3 low).
                     00701 ;* During keyboard scan, LEDs L, H and P are serviced. Logic state at
                     00702 ;* PORTA,4 affects LEDs L and H directly, and LED P is under control
                     00703 ;* of down counter PCOUNT. This counter is initialized at every logic
                     00704 ;* level transition at PORTA,4, and while counting down, if PCOUNT>0,
                     00705 ;* LED P is switched on.
                     00706 ;* Loop labeled "Unstable" adds the extra delay which timing is not
                     00707 ;* constant, but changes from 3 to 49t. This mins the interference
                     00708 ;* between the input scan and tested signal frequency.
                     00709 ;* Counter TMR0 is used for detecting of short pulses. At each
                     00710 ;* transition detected, TMR0 is cleared, then periodicaly tested if
                     00711 ;* counter state was incremented. If so, PCOUNT is initialized and LED
                     00712 ;* P turned on.
                     00713 ;* Register DJNZ is used as a freerunning counter, which divides loop
                     00714 ;* count by 256 and slows down PCOUNT countdown / keys scanning. Keys
                     00715 ;* are debounced and falling edge (pressing moment) detected by
                     00716 ;* rotating registers DEBO1 and DEBO2, and testing them if the key was
                     00717 ;* unpressed at least at 7 passes and then pressed at 1 pass.
                     00718 ;*
                     00719 ;* Input variables: none
                     00720 ;* Output variables: Bit STATUS,C reset if key 1 pressed, set if key 2
                     00721 ;*****************************************************************
0176                 00722 GoKbd
0176  080E           00723         movf   DJNZ,W      ; 1t to avoid intrference,total avg 29t
0177  38F0           00724         iorlw  0f0h        ; 1t extra time range from -.16 and -1
0178  008F           00725         movwf  SCRATCH     ; 1t here SCRATCH varies 0f0h to 0ffh
0179                 00726 UnStable
0179  0F8F           00727         incfsz SCRATCH,F   ; 1-17t (avg .9) adv extra timing count
017A  2979           00728         goto   UnStable    ; 2-32t (avg .17) loop loses extra time
                     00729
017B  0801           00730         movf   TMR0,W      ; TMR0 = hardware transition detector
017C  1903           00731         btfsc  STATUS,Z    ; test if transition at PORTA,4...
017D  2980           00732         goto   NoIniP      ; ...if not, do not affect LED P status
017E  1590           00733         bsf    PCOUNT,3    ; initialize PCOUNT for LED P timing
017F  0181           00734         clrf   TMR0        ; re-init hardware transition detector
0180                 00735 NoIniP
0180  3028           00736         movlw  28h         ; bit 4 (TOSE) RESET: L-to-H transition
0181  1A05           00737         btfsc  PORTA,4     ; if probe tip low,leave TOSE reset...
0182  3038           00738         movlw  38h         ; if high, set TOSE: H-to-L transition
                     00739
0183  1683           00740         bsf    STATUS,RP0  ; select bank 1 of registers
0184  0081           00741         movwf  OPTION_REG  ; set/reset TOSE
0185  1283           00742         bcf    STATUS,RP0  ; reselect bank 0
                     00743
0186  1019           00744         bcf    DELAYH,0    ; init input bit in delay HI rotor
0187  1018           00745         bcf    DELAYL,0    ; init input bit in delay LOW rotor
0188  3004           00746         movlw  4           ; value 4 = bit 2 set
0189  0686           00747         xorwf  PORTB,F     ; change state of PORTB.2 (square wave.
                     00748                            ; generation probe tip hi-imp output
018A  1A05           00749         btfsc  PORTA,4     ; test probe input logic level...
018B  2990           00750         goto   InputHi     ; ...and jump if case 2: input high
                     00751                            ; ...or continue if case 1: input low
018C  1D06           00752         btfss  PORTB,2     ; test hi-impedance output state and...
018D  1418           00753         bsf    DELAYL,0    ; ...turn on led L, hi-ipm out was lo
018E  110C           00754         bcf    FLAG,PTIP   ; reset flag to notify that previous...
                     00755                            ; ...state of probe tip was low
018F  2993           00756         goto   ContInpX    ; jump to skip case 2
0190                 00757 InputHi                     ; entry point for case 2: input high
0190  1906           00758         btfsc  PORTB,2     ; test hi-impedance output state and...
```

```
0191  1419            00759         bsf     DELAYH,0    ; ...turn on led L, PORTB,2 was high
0192  150C            00760         bsf     FLAG,PTIP   ; set flag to notify that previoust...
                      00761                             ; ...state of probe tip was high
0193                  00762 ContInpX                    ; moves LHP leds from FLAG to PORTB
0193  138C            00763         bcf     FLAG,LEDL   ; init input bit for led L delay rotor
0194  0898            00764         movf    DELAYL,F    ; test DEALYL status...
0195  1D03            00765         btfss   STATUS,Z    ; ...and skip if DELAYL=0
0196  178C            00766         bsf     FLAG,LEDL   ; turn on led L if DELAYL rotor > 0
0197  0D98            00767         rlf     DELAYL,F    ; propagate bit thru DELAYL rotor
                      00768
0198  130C            00769         bcf     FLAG,LEDH   ; init input bit for led H delay rotor
0199  0899            00770         movf    DELAYH,F    ; test DEALYH status...
019A  1D03            00771         btfss   STATUS,Z    ; ...and skip if DELAYH=0
019B  170C            00772         bsf     FLAG,LEDH   ; turn on led H if DELAYH rotor > 0
019C  0D99            00773         rlf     DELAYH,F    ; propagate bit thru DELAYH rotor
                      00774
019D  23F0            00775         call    MoveLESd    ; send leds status flag bits to PORTB
                      00776
019E  0F8E            00777         incfsz  DJNZ,F      ; test if this is 256th pass...
019F  2976            00778         goto    GoKbd       ; ...if not, loop
                      00779
                      00780 ;------------------------ passes here each 256 pass (about 8.9ms)
                      00781
01A0  128C            00782         bcf     FLAG,LEDP   ; reset led P flag (set if PCOUNT>0)
01A1  0890            00783         movf    PCOUNT,F    ; test PCOUNT status...
01A2  1903            00784         btfsc   STATUS,Z    ; ...and skip jump if PCOUNT>0
01A3  29A9            00785         goto    PCount0     ; ...else jump if PCOUNT = 0
                      00786
01A4  0390            00787         decf    PCOUNT,F    ; if PCOUNT>0, then decrement it
01A5  0818            00788         movf    DELAYL,W    ; W>0 if led L is on
01A6  0419            00789         iorwf   DELAYH,W    ; W>0 if led L or led H is on
01A7  1D03            00790         btfss   STATUS,Z    ; ...skip if both L and H leds are off
01A8  168C            00791         bsf     FLAG,LEDP   ; if PCOUNT>0, dec PCOUNT, & set led P
01A9                  00792 PCount0
                      00793                             ; ----------- key 2 test (right key)
01A9  0E05            00794         swapf   PORTA,W     ; let key 1&2 status move to bits 6&7
01AA  008F            00795         movwf   SCRATCH     ; SCRATCH,7=key2, SCARTCH,6=key1
01AB  098F            00796         comf    SCRATCH,F   ; complement to set bit if key pressed
                      00797
01AC  0D8F            00798         rlf     SCRATCH,F   ; set C if key 2 pressed
                      00799
01AD  0D93            00800         rlf     DEBO2,F     ; propagate key 2 bit thru rotor
01AE  1003            00801         bcf     STATUS,C    ; reset C,notify at exit key 2 pressed
01AF  0313            00802         decf    DEBO2,W     ; DEBO2 = b'00000001' if just pressed
01B0  1903            00803         btfsc   STATUS,Z    ; skip return if key 2 not just pressed
01B1  0008            00804         return              ; *** exit 1: key 2 just pressed (NC)
                      00805
                      00806                             ; ----------- key 1 test (left key)
                      00807
01B2  0D8F            00808         rlf     SCRATCH,F   ; set C if key 1 pressed
                      00809
01B3  0D92            00810         rlf     DEBO1,F     ; propagate key 1 bit thru rotor
01B4  1403            00811         bsf     STATUS,C    ; set C, notify at exit key 1 pressed
01B5  0312            00812         decf    DEBO1,W     ; DEBO1 = b'00000001' if just pressed
01B6  1903            00813         btfsc   STATUS,Z    ; skip return if key 1 not just pressed
01B7  0008            00814         return              ; *** exit 2: key 1 just pressed (C)
                      00815
01B8  0B9B            00816         decfsz  TIMOUTL,F   ; timeout lo counter...
01B9  2976            00817         goto    GoKbd       ; ... not yet zero: loop
                      00818
01BA  0B9C            00819         decfsz  TIMOUTH,F   ; timeout hi counter...
01BB  2976            00820         goto    GoKbd       ; ... not yet zero: loop
                      00821                             ; *** exit 3:cont with timeout process
                      00822
                      00823 ;*****************************************************************
                      00824 ;* Power Off entry point
```

```
                    00825 ;* Wait until both keys off for 34 ms, and then switch power off.
                    00826 ;* PORTB,3, when low, switches the unit off.
                    00827 ;*****************************************************************
01BC                00828 Suicide
01BC  21BF          00829         call    KeysOff     ; test keys off to avoud re-trigering
01BD  0186          00830         clrf    PORTB       ; pull PORTB,3 low to switch power off
01BE  29BE          00831         goto    $           ; loop until power off
                    00832
                    00833 ;*****************************************************************
                    00834 ;* KeysOff
                    00835 ;* Loop until both keys off for 34 ms, then exit
                    00836 ;*
                    00837 ;* Input variables: none
                    00838 ;* Output variables: TIMOUTH is cleared to 0
                    00839 ;*****************************************************************
01BF                00840 KeysOff
01BF  019C          00841         clrf    TIMOUTH     ; initialize pointer
01C0                00842 BothOff
01C0  1905          00843         btfsc   PORTA,2     ; skip if key 1 on
01C1  1D85          00844         btfss   PORTA,3     ; do not skip if key 2 on
01C2  29BF          00845         goto    KeysOff     ; reinitialize ptr if any key on
01C3  23D0          00846         call    loop130     ; loop 130us
01C4  0B9C          00847         decfsz  TIMOUTH,F   ; test pointer
01C5  29C0          00848         goto    BothOff     ; loop 256xs to verify both keys off
01C6  0008          00849         return              ; both keys are off for at least 34 ms
                    00850
                    00851 ;*******  ANALYZER
                    00852 ;*****************************************************************
                    00853 ;* Pointer2
                    00854 ;* Writes 2 measuring points for analyzer reference. 1st=TIMOUTL*10+5,
                    00855 ;* then TIMOUTL is incremented by 2 and the second one is TIMOUTL*10+0
                    00856 ;*
                    00857 ;* Input variables: TIMOUTL will first be printed as TIMOUTL*10+5
                    00858 ;* Output variables: TIMOUTL incremented by 3 (pointer advanced by 30)
                    00859 ;*****************************************************************
01C7                00860 Pointer2
01C7  21CF          00861         call    Pointer1    ; first two digits
01C8  3035          00862         movlw   '5'         ; third digit is 5 for odd pointer
01C9  0A9B          00863         incf    TIMOUTL,F   ; each incr advances pointer by 10
01CA  21CD          00864         call    AdvToCh     ; advance ptr and print "5"
01CB  21CF          00865         call    Pointer1    ; first two digits
01CC  304F          00866         movlw   'O'         ; third digit is 0 for even pointer
01CD                00867 AdvToCh
01CD  0A9B          00868         incf    TIMOUTL,F   ; each incr advances pointer by 10
01CE  2BD8          00869         goto    Char        ; print 0 or 5
                    00870
                    00871 ;*****************************************************************
                    00872 ;* Pointer1
                    00873 ;* Writes blank, then symbol "^" and then converts TIMOUTL and prints
                    00874 ;* as 2 digits.
                    00875 ;*
                    00876 ;* Input variables: TIMOUTL, bin value which prints as 2-digits
                    00877 ;* Output variables: none
                    00878 ;*****************************************************************
01CF                00879 Pointer1
01CF  23D7          00880         call    Blank       ; skip first 3 samples (one character)
01D0  305E          00881         movlw   '^'         ; "^" (pointing tool)
01D1                00882 PrintTL
01D1  23D8          00883         call    Char        ; print "^"
01D2  081B          00884         movf    TIMOUTL,W   ; TIMOUTL is the main pointer for...
01D3  2B70          00885         goto    Print255    ; ...first two digits
                    00886
                    00887 ;*****************************************************************
                    00888 ;* Mode1Show
                    00889 ;* Write measuring points at row 2, and wait until keys are released.
                    00890 ;* Then incr. SHOWCOU in range 0...4 and continue to Draw subroutine
```

```
                        00891 ;* This command,which executes when key 2 is pressed in analyzer mode,
                        00892 ;* while the cursor is on the number of 60-sample group, advances the
                        00893 ;* pointer.
                        00894 ;* It continues to Draw routine.
                        00895 ;*************************************************************************
01D4                    00896 Mode1Show
01D4  23CE              00897         call    Row2        ; pointers are in row 2
01D5  1003              00898         bcf     STATUS,C    ; prepare for multiplying by 2
01D6  0D17              00899         rlf     SHOWCOU,W   ; W=SHOWCOU*2
01D7  009B              00900         movwf   TIMOUTL     ; TIMOUTL=SHOWCOU*2
01D8  0D9B              00901         rlf     TIMOUTL,F   ; TIMOUTL-SHOWCOU*4
01D9  079B              00902         addwf   TIMOUTL,F   ; TIMOUTL-SHOWCOU*6
                        00903
01DA  21C7              00904         call    Pointer2    ; print 1st and 2nd pointer
01DB  21C7              00905         call    Pointer2    ; print 3rd and 4th pointer
                        00906
01DC  21BF              00907         call    KeysOff     ; wait until key off
01DD  227D              00908         call    PrintM1     ; restore normal row 2
                        00909
01DE  0A97              00910         incf    SHOWCOU,F   ; advance number of groups displayed
01DF  1917              00911         btfsc   SHOWCOU,2   ; test if SHOWCOU=5: first test bit2...
01E0  1C17              00912         btfss   SHOWCOU,0   ; test if SHOWCOU=5: ...then test bit 0
01E1  29E3              00913         goto    Draw        ; skip wrapping if SHOWCOU<5
01E2  0197              00914         clrf    SHOWCOU     ; SHOWCOU cycle in range 0...4
                        00915
                        00916 ;*************************************************************************
                        00917 ;* Draw
                        00918 ;* This subroutine writes 20 pseudographic chars in line 1 in analyzer
                        00919 ;* mode. First, whole buffer is rotated 0/60/120/180/240 bit places to
                        00920 ;* the right (if SHOWCOU=0/1/2/3/4,respectively) to adj. the sequence
                        00921 ;* to be displayed to the start of the buffer. Then the string of 20
                        00922 ;* 3-bit groups is rotated right, and displayed as 20 special chars
                        00923 ;* (codes 0-7), defined at program setup (at loop labeled GoGraph).
                        00924 ;* Then the buffer is rotated again, to the total of 304 bit places,
                        00925 ;* so the buffer contents is unmodified on exit.
                        00926 ;*
                        00927 ;* Input variables:
                        00928 ;* SHOWCOU, denotes which group of 60 samples will be displayed
                        00929 ;* Output variables: none
                        00930 ;*************************************************************************
01E3                    00931 Draw
01E3  0817              00932         movf    SHOWCOU,W   ; prep to rotate buf SHOWCOU*60 times
01E4  1D03              00933         btfss   STATUS,Z    ; avoid rotating if SHOWCOU=0
01E5  21F9              00934         call    Carusel     ; rotate buffer SHOWCOU*60 times
                        00935
01E6  22BF              00936         call    Row1        ; samples must be written in row 1
01E7  3014              00937         movlw   .20         ; .20 characters to write
01E8  0096              00938         movwf   CHARCOU     ; CHARCOU is the main character counter
01E9                    00939 Go20Chars
01E9  018F              00940         clrf    SCRATCH     ; register for 3-bit code gen (0...7)
01EA  2202              00941         call    RRBuf       ; bit from buffer in C...
01EB  0D8F              00942         rlf     SCRATCH,F   ; ...bit from C in code register
01EC  2202              00943         call    RRBuf       ; bit from buffer in C...
01ED  0D8F              00944         rlf     SCRATCH,F   ; ...bit from C in code register
01EE  2202              00945         call    RRBuf       ; bit from buffer in C...
01EF  0D0F              00946         rlf     SCRATCH,W   ; ...bit from C in code reg and in W
01F0  23D9              00947         call    CharNCC     ; draw one char = 3 samples
01F1  0B96              00948         decfsz  CHARCOU,F   ; 20 characters written?
01F2  29E9              00949         goto    Go20Chars   ; no, loop
                        00950
01F3  0817              00951         movf    SHOWCOU,W   ; prep to rotate to total of 304 bits
01F4  3C04              00952         sublw   .4          ; W=4-SHOWCOU
01F5  1D03              00953         btfss   STATUS,Z    ; avoid rotating if SHOWCOU=0
01F6  21F9              00954         call    Carusel     ; rotate buffer again to restore it
01F7  2200              00955         call    RRBuf4      ; four more times to get 304 times
01F8  28CD              00956         goto    Farm1       ; done, go back to user interface
```

```
                    00957
                    00958 ;*******************************************************************
                    00959 ;* Carusel
                    00960 ;* This subroutine rotates BUFFER right W*60 times
                    00961 ;* Note: if W=0, rotating will be performed 1024 times
                    00962 ;*
                    00963 ;* Input variables: W,how many (*60) x the buf is rotated right (W>0)
                    00964 ;* Output variables: none
                    00965 ;*******************************************************************
01F9                00966 Carusel
01F9  008F          00967         movwf   SCRATCH     ; SCRATCH=W
01FA  0E8F          00968         swapf   SCRATCH,F   ; SCRATCH=W*16
01FB  028F          00969         subwf   SCRATCH,F   ; SCRATCH=W*15
01FC                00970 RRLoop
01FC  2200          00971         call    RRBuf4      ; 4 rotates in every pass
01FD  0B8F          00972         decfsz  SCRATCH,F   ; done W*15*4 times?
01FE  29FC          00973         goto    RRLoop      ; no, continue rotating BUFFER
01FF  0008          00974         return              ; done
                    00975
                    00976 ;*******************************************************************
                    00977 ;* RRBuf4 executes RRBuf 4 times
                    00978 ;* RRBuf rotates buffer(38 bytes=304 bits) right for one bit position.
                    00979 ;* Bit STATUS,C is first loaded from the first bit in buffer, so
                    00980 ;* will rotating be completely performed at 304 bits, not through C.
                    00981 ;*
                    00982 ;* Input/Output variables: none
                    00983 ;*******************************************************************
0200                00984 RRBuf4                      ; 4 times rotates right 38 bytes
0200  2201          00985         call    RRBuf2      ; rotate BUFFER 2* and then again 2*
0201                00986 RRBuf2
0201  2202          00987         call    RRBuf       ; rotate BUFF once and then again once
0202                00988 RRBuf                       ; rotates rt 38 bytes,bit in C at exit
0202  304B          00989         movlw   BUFFER+.37  ; start with last byte to be rotated
0203  0084          00990         movwf   FSR         ; FSR = pointing register for rotating
0204  3026          00991         movlw   .38         ; 38 bytes total buffer
0205  0094          00992         movwf   COUNT       ; byte counter
                    00993
0206  1003          00994         bcf     STATUS,C    ; C rotated into BUFFER+.37,so it...
0207  1826          00995         btfsc   BUFFER,0    ; ...must be equal to bit BUFFER+0,0...
0208  1403          00996         bsf     STATUS,C    ; ...to perform non-destruct rotating
0209                00997 ByteLoop
0209  0C80          00998         rrf     INDF,F      ; byte rotated here
020A  0384          00999         decf    FSR,F       ; let FSR point to next byte
020B  0B94          01000         decfsz  COUNT,F     ; COUNT is byte counter, done?
020C  2A09          01001         goto    ByteLoop    ; no, loop
020D  0008          01002         return              ; here the output bit is in STATUS,C
                    01003
                    01004 ;*******************************************************************
                    01005 ;* Mode1Go
                    01006 ;* This is entry point for analyzer start command (symbol * ). After
                    01007 ;* clearing line1 of LCD, the program vectors to routines which handle
                    01008 ;* different sampling rates. Three highest rates (1 MHz, 500 KHz and
                    01009 ;* 228 KHz are sampled at individual routines (Get1MHz, Get500KHz and
                    01010 ;* Get228KHz), andthe remining rates at routine GetSlowClk. All those
                    01011 ;* routines (except Get1MHz, which is location-sensitive (so it is at
                    01012 ;* the very beginning of the program), are listed here.
                    01013 ;*
                    01014 ;* Input/Output variables: none
                    01015 ;*******************************************************************
020E                01016 Mode1Go                     ; *** ept: analyzer start
                    01017
020E  22B7          01018         call    ClrRow1     ; clear LCD line 1 and turn LEDs off
                    01019
020F  0815          01020         movf    RATE,W      ; RATE = sample rate in range 0...15
0210  1903          01021         btfsc   STATUS,Z    ; skip if sample rate>0
0211  2801          01022         goto    Get1MHz     ; jump to individual routine if RATE=0
```

```
                    01023
                    01024                               ; --- try 500 KHz rate
0212  008F           01025          movwf   SCRATCH      ; SCRATCH = RATE
0213  0B8F           01026          decfsz  SCRATCH,F    ; test if RATE=1
0214  2A2B           01027          goto    Try228KHz    ; if not RATE=1, then try if RATE=2
                    01028                               ; if RATE=1, then drop to Get500KHz
                    01029
                    01030 ;*****************************************************************
                    01031 ;* Get500KHz
                    01032 ;* This subroutine fetches 304 samples at 2us rate (5 instr cycles
                    01033 ;* timing).
                    01034 ;* Call Common initializes loop counter (COUNT) to 38*8=304 samples
                    01035 ;* and FSR to point to BUFFER. It also presets T0SE bit depended on
                    01036 ;* XTOX bit (in FLAG register)to enable proper edge detecting, as it
                    01037 ;* will affect TMR0 state.
                    01038 ;* State of key 2(Break) tested while waiting for starting condition.
                    01039 ;*
                    01040 ;* Input/Output variables: none
                    01041 ;*****************************************************************
0215                01042 Get500KHz                     ; 5 t read cycle
0215  3026           01043          movlw   .38          ; 38 bytes * 8 bits = 304 samples
0216  226F           01044          call    Common       ; initialize COUNT, FSR, hi-imp out...
                    01045                               ; ...bit XTOX and T0SE bit
0217                01046 Edge500
0217  1D85           01047          btfss   PORTA,3      ; test status of key 2 and...
0218  28CC           01048          goto    Break        ; ...jump to Break routine if pressed
                    01049
0219  0801           01050          movf    TMR0,W       ; TMR0 = logic level edge detector
021A  1903           01051          btfsc   STATUS,Z     ; test if there was egde...
021B  2A17           01052          goto    Edge500      ; ...and loop if not
021C  0384           01053          decf    FSR,F        ; adj pointer as it will be advanced...
                    01054                               ; ... before data write
021D                01055 Get500Loop
021D  0C85           01056          rrf     PORTA,F      ; <-- move input status to C
021E  0A84           01057          incf    FSR,F        ; advance write pointer
021F  0C80           01058          rrf     INDF,F       ; write bit C in destination rotor
0220  3006           01059          movlw   .6           ; initialize count for 6 bits
0221  008E           01060          movwf   DJNZ         ; DJNZ = bit counter
0222                01061 Go6Bits
0222  0C85           01062          rrf     PORTA,F      ; <-- 6*  move input status to C
0223  0C80           01063          rrf     INDF,F       ; write bit C in destination rotor
0224  0B8E           01064          decfsz  DJNZ,F       ; DJNZ = bit counter
0225  2A22           01065          goto    Go6Bits      ; loop if not yet 6 bits fetched
                    01066
0226  0C85           01067          rrf     PORTA,F      ; <--     move input status to C
0227  0C80           01068          rrf     INDF,F       ; write bit C in destination rotor
0228  0B94           01069          decfsz  COUNT,F      ; COUNT = byte counter
0229  2A1D           01070          goto    Get500Loop   ; loop if not yet 38 bytes fetched
022A  2A6A           01071          goto    Finished     ; all bits fetched; go display them
                    01072
                    01073 ;*****************************************************************
                    01074 ;* Test if register SCRATCH reaches 0 after decrementing(this happens
                    01075 ;* if RATE = 2), if so drop to Get228KHz else go to GetSlowClk
                    01076 ;*****************************************************************
022B                01077 Try228KHz
022B  0B8F           01078          decfsz  SCRATCH,F    ; test RATE status (SCRATCH=RATE-1)
022C  2A3F           01079          goto    GetSlowClk   ; jump if not RATE=2
                    01080
                    01081 ;*****************************************************************
                    01083 ;* This subroutine fetches 304 samples at 4.4us rate (11 instruction
                    01084 ;* cycles timing).
                    01085 ;* Call Common304 initializes loop counter (COUNT) to 304 samples
                    01086 ;* and FSR to point to BUFFER. It also presets T0SE bit depended on
                    01087 ;* XTOX bit (in FLAG register)to enable proper edge detecting, as it
                    01088 ;* will affect TMR0 state.
                    01089 ;* State of key 2(Break) tested while waiting for starting condition.
```

```
                     01090 ;*
                     01091 ;* Input/Ouput variables: none
                     01092 ;*****************************************************************
022D                 01093 Get228KHz                ; 11 t read cycle
022D  226C           01094         call    Common304   ; init COUNT(lo), TIMOUTH(hi byte)...
                     01095                             ; ...FSR, hi-imp outbit XTOX & T0SE bit
022E                 01096 Edge228
022E  1D85           01097         btfss   PORTA,3     ; test status of key 2 and...
022F  28CC           01098         goto    Break       ; ...jump to Break routine if pressed
                     01099
0230  0801           01100         movf    TMR0,W      ; TMR0 = logic level edge detector
0231  1903           01101         btfsc   STATUS,Z    ; test if there was egde...
0232  2A2E           01102         goto    Edge228     ; ...and loop if not
0233                 01103 Go228
0233  2A34           01104         goto    $+1         ; 2     two extra cycles to make 11t
0234                 01105 Go228B
0234  0C85           01106         rrf     PORTA,F ; <---  ; 1     move input bit to C
0235  0C80           01107         rrf     INDF,F      ; 1     write bitC in destination rotor
                     01108
0236  0314           01109         decf    COUNT,W     ; 1     COUNT = bit counter
0237  3907           01110         andlw   7           ; 1     test if 8th pass...
0238  1903           01111         btfsc   STATUS,Z    ; 1 (2) ...and skip if not
0239  0A84           01112         incf    FSR,F       ; 1 (0) ...else advance pointer
                     01113
023A  0B94           01114         decfsz  COUNT,F     ; 1     COUNT = (lo byte) bit counter
023B  2A33           01115         goto    Go228       ; 2     loop if not yet = 0
                     01116
023C 0B9C            01117         decfsz  TIMOUTH,F       ; TIMOUTH = (hi byte) bit counter
023D 2A34            01118         goto    Go228B          ; this does not add extra cycles, as it
                     01119                                 ; ...jumps after goto $+1
023E 2A6A            01120         goto    Finished        ; all bits fetched; go display them
                     01121
                     01122 ;*****************************************************************
                     01123 ;* GetSlowClk
                     01124 ;* This subroutine fetches 304 samples at variable rate, depended on
                     01125 ;*  RATE (SCRATCH=RATE-3). Timing constant is loaded from lookup table
                     01126 ;*  located at DATA EEPROM (locations 30h-3ch).
                     01127 ;*
                     01128 ;* Call Common304 initializes 16-bit loop counter to 304 samples
                     01129 ;*  (lo byte: COUNT=.304-.256, hi byte: TIMOUTH=.1)
                     01130 ;*  and FSR to point to BUFFER. It also presets T0SE bit depended on
                     01131 ;*  XTOX bit (in FLAG register)to enable proper edge detecting, as it
                     01132 ;*  will affect TMR0 state.
                     01133 ;*
                     01134 ;* Rates 3-11 (100KHz-2.4KHz) have loop period of factor from EEPROM
                     01135 ;*  table multiplied by 5 instruction cycles and adding 20 instruction
                     01136 ;*  cycles (Factor*5T+20T), and rates 12-15 (1.2KHz-40Hz) multilied the
                     01137 ;*  factor by 417 instruction cycles and adding 415 instruction cycle
                     01138 ;*  (Factor*417T+415T)
                     01139 ;*
                     01140 ;*  RATE =  3, factor:   1, T/cycle:    25, Freq:  100   KHz
                     01141 ;*  RATE =  4, factor:   6, T/cycle:    50, Freq:   50   KHz
                     01142 ;*  RATE =  5, factor:   9, T/cycle:    65, Freq:   38.4 KHz
                     01143 ;*  RATE =  6, factor:  16, T/cycle:   100, Freq:   25   KHz
                     01144 ;*  RATE =  7, factor:  22, T/cycle:   130, Freq:   19.2 KHz
                     01145 ;*  RATE =  8, factor:  46, T/cycle:   250, Freq:   10   KHz
                     01146 ;*  RATE =  9, factor:  48, T/cycle:   260, Freq:    9.6 KHz
                     01147 ;*  RATE = 10, factor: 100, T/cycle:   520, Freq:    4.8 KHz
                     01148 ;*  RATE = 11, factor: 204, T/cycle:  1040, Freq:    2.4 KHz
                     01149 ;*  RATE = 12, factor:   5, T/cycle:  2500, Freq:    1   KHz
                     01150 ;*  RATE = 13, factor:  14, T/cycle:  6253, Freq:  400    Hz
                     01151 ;*  RATE = 14, factor:  59, T/cycle: 25018, Freq:  100    Hz
                     01152 ;*  RATE = 15, factor: 149, T/cycle: 62548, Freq:   40    Hz
                     01153 ;*
                     01154 ;* State of key 2 (Break) is tested while waiting for start condition.
                     01155 ;* LED P is turned on while sampling, indicate sample period at slower
```

```
                    01156 ;*  rates, in which it appears to be visible.
                    01157 ;*
                    01158 ;* Input variables: RATE, affects timing factor
                    01159 ;* Output variables: none
                    01160 ;************************************************************************
023F                01161 GetSlowClk
023F 030F           01162         decf    SCRATCH,W       ; W = RATE-3
0240 3E30           01163         addlw   .48             ; rate table in data eeprom @ addr .48
                    01164
0241 22AC           01165         call    AGet_EE         ; get time const. from dataeeprom table
0242 008F           01166         movwf   SCRATCH         ; time const for rates 3-15 -> SCRATCH
                    01167
0243 226C           01168         call    Common304       ; init COUNT(lo), TIMOUTH(hi byte)...
                    01169                                 ; ...FSR, hi-imp outbit XTOX & T0SE bit
0244                01170 EdgeSlow
0244 1D85           01171         btfss   PORTA,3         ; test status of key 2 and...
0245 28CC           01172         goto    Break           ; ...jump to Break routine if pressed
                    01173
0246 0801           01174         movf    TMR0,W          ; TMR0 = logic level edge detector
0247 1903           01175         btfsc   STATUS,Z        ; test if there was egde...
0248 2A44           01176         goto    EdgeSlow        ; ...and loop if not
                    01177
0249 1686           01178         bsf     PORTB,5         ; turn led P on, notify sampling on
                    01179
024A                01180 GoSlow
024A 0C85           01181         rrf     PORTA,F ; <---  ; 1     move input bit to C
024B 0C80           01182         rrf     INDF,F          ; 1     write bitC in destination rotor
                    01183
024C 0314           01184         decf    COUNT,W         ; 1     COUNT = bit counter
024D 3907           01185         andlw   7               ; 1     test if 8th pass...
024E 1903           01186         btfsc   STATUS,Z        ; 1 (2) ...and skip if not
024F 0A84           01187         incf    FSR,F           ; 1 (0) ...else advance pointer
                    01188
0250 1995           01189         btfsc   RATE,3          ; 1 1   if bits2 and 3 cleared, that...
0251 1D15           01190         btfss   RATE,2          ; 2 1   ...means that rate<12...
0252 2A62           01191         goto    Not417          ; - 2   ...if so, jump to short timing
                    01192                                 ; --------417 (RATE 12-15,bits 2,3 set)
0253 080F           01193         movf    SCRATCH,W       ; 1       SCRATCH = timing constant
0254 0096           01194         movwf   CHARCOU         ; 1       CHARCOU = loop counter
0255                01195 Loop417                         ;   \
0255 3052           01196         movlw   .82             ; 1  \
0256 008E           01197         movwf   DJNZ            ; 1   \
0257 23D3           01198         call    Loop7           ; 411  > total cyc here 417*SCRATCH-1
0258 0000           01199         nop                     ; 1   /
0259 0B96           01200         decfsz  CHARCOU,F       ; 1  / 2t at exit only
025A 2A55           01201         goto    Loop417         ; 2 /  0t at exit only
                    01202
025B 0806           01203         movf    PORTB,W         ; 1
025C 39DF           01204         andlw   0dfh            ; 1     reset bit 5 (LED P)
025D 1E14           01205         btfss   COUNT,4         ; 1 2
025E 3820           01206         iorlw   20h             ; 1 0  set bit 5 (LED P) if COUNT,4 = 0
025F 0086           01207         movwf   PORTB           ; 1    blink LED P while sampling each
                    01208                                 ;       16th pass
0260 304E           01209         movlw   .78             ; 1    constant for long timing
0261 2A64           01210         goto    SameAs5         ; 2    skip short timing
0262                01211 Not417                          ; ---------- 5 (RATE 3-11)
0262 2A63           01212         goto    $+1             ; 2    waist 2 cycles
0263 080F           01213         movf    SCRATCH,W       ; 1    SCRATCH = timing constant
0264                01214 SameAs5                         ; ---------- 417 and 5 (RATE 3-15)
0264 008E           01215         movwf   DJNZ            ; 1    DJNZ = loop counter
0265 23D2           01216         call    GoLoop          ; short time: 3T+5T*SCRATCH, long:393T
                    01217
0266 0B94           01218         decfsz  COUNT,F         ; 1    COUNT = (lo byte) bit counter
0267 2A4A           01219         goto    GoSlow          ; 2    loop if not yet = 0
                    01220
0268 0B9C           01221         decfsz  TIMOUTH,F       ; TIMOUTH = (hi byte) bit counter
```

```
0269 2A4A          01222         goto    GoSlow          ; adds 2T extra 1time, after 48th pass
026A               01223 Finished
026A 0197          01224         clrf    SHOWCOU         ; reset pointer to 1st group of 60 samp
026B 29E3          01225         goto    Draw            ; all bits fetched; display them
                   01226
                   01227 ;*****************************************************************
                   01228 ;* Common304
                   01229 ;* This subroutine initializes low byte loop counter (COUNT) to 304
                   01230 ;*  samples (lo byte: COUNT=.304-.256, hi byte: TIMOUTH=.1+1)
                   01231 ;*  and FSR to point to BUFFER. It also presets T0SE bit depended on
                   01232 ;*  XTOX bit (in FLAG register)to enable proper edge detecting, as it
                   01233 ;*  will affect TMR0 state.
                   01234 ;* Entry point Common allows subroutines Get1MHz and GetSlowClk to
                   01235 ;*  preset COUNT to another values.
                   01236 ;*
                   01237 ;* Input variables:
                   01238 ;*    For entry point COMMON, register W is placed in COUNT
                   01239 ;* Output variables:
                   01240 ;*    COUNT is initialized to # of loop passes (W or low byte of 304)
                   01241 ;*    TMR0 is cleared
                   01242 ;*    T0SE and PORTB,2 are copied from FLAG,XTOX
                   01243 ;*****************************************************************
026C               01244 Common304
026C 3002          01245         movlw   .2              ; hi byte=2 for reg lo byte value...
                   01246                                 ; ...plus extra 256 passes
026D 009C          01247         movwf   TIMOUTH         ; hi byte loop counter for 304 passes
026E 3030          01248         movlw   .304-.256       ; lo byte value for 304 passes
026F               01249 Common
026F 0094          01250         movwf   COUNT           ; COUNT = loop counter
0270 3026          01251         movlw   BUFFER          ; first byte of destination
0271 0084          01252         movwf   FSR             ; FSR = destination pointer
                   01253
0272 3028          01254         movlw   28h             ; initialize T0SE fot L-to-H transition
0273 1106          01255         bcf     PORTB,2         ; clr hi-imp out if expecting rise edge
                   01256
0274 1A0C          01257         btfsc   FLAG,XTOX       ; test slctd edge for start condition
0275 2A78          01258         goto    ToOption        ; and skip falling edge if rise slctd
                   01259
0276 3038          01260         movlw   38h             ; initialize T0SE fot H-to-L transition
0277 1506          01261         bsf     PORTB,2         ; set hi-imp out if expecting fall edge
0278               01262 ToOption
0278 1683          01263         bsf     STATUS,RP0      ; select bank 1 of registers
0279 0081          01264         movwf   OPTION_REG      ; set/reset TOSE
027A 1283          01265         bcf     STATUS,RP0      ; reselect bank 0
027B 0181          01266         clrf    TMR0            ; initialize TMR0 as edge detector
027C 0008          01267         return                  ; finished
                   01268
                   01269 ;*****************************************************************
                   01270 ;* PrintM1
                   01271 ;* Print string at line 2 in analyzer mode.
                   01272 ;* At pos 0, rate in format XX[[.]X][M|K]Hz/XX[[.]X][u|m]s is printed.
                   01273 ;*  Those values picked from table located in Data EEPROM, locations
                   01274 ;*  0-2Fh. Register CHARCOU is used to fill blanks up to pos 13 in line
                   01275 ;*  2, to disable phantom characters appearance when changing rate from
                   01276 ;*  some long-string to short-string value.
                   01277 ;* Symbol for starting or rising edge for starting event is written at
                   01278 ;*  pos 13, symbol "*" for start command at pos 15 and the number of
                   01279 ;*  group displayed at pos. 17.
                   01280 ;*
                   01281 ;* Input variables:
                   01282 ;*    RATE will be printed in pos 0, row2
                   01283 ;*    bit FLAG,XTOX affects the printed symbol of rising/falling edge
                   01284 ;*    SHOWCOU (number of group displayed) is printed as numeric (+1)
                   01285 ;* Output variables: none
                   01286 ;*****************************************************************
027D               01287 PrintM1
```

```
027D 23CE          01288         call    Row2            ; move cursor to row 2
                   01289
027E 300D          01290         movlw   .13             ; init counter for 13 char fix format
027F 0096          01291         movwf   CHARCOU         ; CHARCOU = character counter
                   01292
0280 0815          01293         movf    RATE,W          ; W = RATE
0281 0715          01294         addwf   RATE,W          ; W = 2 * RATE
0282 0715          01295         addwf   RATE,W          ; W = 3 * RATE (ea rate has 3 bytes
                   01296                                 ;                 in table)
0283 22AC          01297         call    AGet_EE         ; get 1st byte via table in data eeprom
0284 009B          01298         movwf   TIMOUTL         ; TIMOUTL = 1st byte from table
0285 0E1B          01299         swapf   TIMOUTL,W       ; move to bits 0-3, bits 4-7 are freq
                   01300
0286 22AD          01301         call    Get_EE          ; get 2nd byte via table in data eeprom
                   01302
0287 1B1B          01303         btfsc   TIMOUTL,6       ; bit 6 = decimal point for frequency
0288 148C          01304         bsf     FLAG,DP         ; set decimal point bit if bit 6 set
                   01305
0289 2373          01306         call    Print3          ; display sampling frequency
                   01307
028A 304D          01308         movlw   'M'             ; for "MHz" display
028B 0895          01309         movf    RATE,F          ; if RATE=0...
028C 1D03          01310         btfss   STATUS,Z        ; ...then let it be MHz
028D 304B          01311         movlw   'K'             ; for "KHz" display
028E 1B9B          01312         btfsc   TIMOUTL,7       ; bit 7=KHz or MHz,skip 1st char if clr
028F 23D8          01313         call    Char            ; print "M" or "K" if bit 7 set
0290 3070          01314         movlw   TxtHz-1+1       ; for "Hz" display
0291 23DD          01315         call    Write           ; print "Hz"
                   01316
0292 081B          01317         movf    TIMOUTL,W       ; TIMOUTL = 1st byte from table
                   01318
0293 22AD          01319         call    Get_EE          ; get 2nd byte via table in data eeprom
                   01320
0294 191B          01321         btfsc   TIMOUTL,2       ; bit 2 = decimal point for period
0295 148C          01322         bsf     FLAG,DP         ; set decimal point bit if bit 2 set
                   01323
0296 2373          01324         call    Print3          ; display digits for period
                   01325
0297 30E4          01326         movlw   0e4h            ; Greek "micro"
0298 1D9B          01327         btfss   TIMOUTL,3       ; if bit 3 set, let it be "micro"
0299 306D          01328         movlw   'm'             ; ...if not, convert to "milli" (m)
029A 23D8          01329         call    Char            ; print "micro" or "m"
029B 3073          01330         movlw   's'             ; s stands for seconds
029C 23D8          01331         call    Char            ; print "s"
029D               01332 XtraChar                       ; adds extra (CHARCOU) blanks
029D 3020          01333         movlw   ' '             ; print blank to overprint prev string
029E 23D9          01334         call    CharNCC         ; print blank without affecting CHARCOU
029F 0B96          01335         decfsz  CHARCOU,F       ; CHARCOU=character counter
02A0 2A9D          01336         goto    XtraChar        ; loop if not yet pos 13
                   01337
                   01338                                 ; ----- here is start condition symbol
02A1 3001          01339         movlw   1               ; symbol of rising edge
02A2 1E0C          01340         btfss   FLAG,XTOX       ; let it be rising if XTOX set
02A3 3004          01341         movlw   4               ; symbol of falling edge
02A4 23D6          01342         call    CharBl          ; print symbol and blank
02A5               01343 KaoAna                         ; ----- here is "start" ("*") symbol
02A5 302A          01344         movlw   '*'             ; "start" symbol
02A6 23D6          01345         call    CharBl          ; print "start" symbol and blank
                   01346                                 ; --- here follows # of 60 smples group
02A7 0A17          01347         incf    SHOWCOU,W       ; SHOWCOU =  number of 60 samples group
02A8 23C9          01348         call    Num             ; print it (incremented by 1)
02A9 23D7          01349         call    Blank           ; print blank
02AA               01350 Arrow
02AA 307E          01351         movlw   7eh             ; 7Eh=right arrow in LM032L char set
02AB 2BD8          01352         goto    Char            ; print arrow in rightmost pos
                   01353
```

```
                    01354 ;*********************************************************************
                    01355 ;* AGet_EE
                    01356 ;* Get_EE
                    01357 ;*  This is routine for reading from Data EEPROM. Writing to BIN4+0 and
                    01358 ;*  BIN4+1 is also integrated here, as those variables are used for bin
                    01359 ;*  to decimal conversion.
                    01360 ;*
                    01361 ;* Input variables: W, data address at AGet_EE
                    01362 ;* Output variables: BIN4, binary data of rate display from DATA EEPROM
                    01363 ;*********************************************************************
02AC                01364 AGet_EE
02AC 0089           01365         movwf   EEADR           ; initialize eeprom address pointer
02AD                01366 Get_EE
02AD 3903           01367         andlw   3               ; hi byte BIN4 for freq display
02AE 009F           01368         movwf   BIN4+1          ; BIN4+1 = hi byte for range 0...999
                    01369
02AF 1683           01370         bsf     STATUS,RP0      ; select bank 1 of registers
02B0 1408           01371         bsf     EECON1,RD       ; set handshaking bit for data ee read
02B1 1283           01372         bcf     STATUS,RP0      ; reselect bank 0
02B2 0808           01373         movf    EEDATA,W        ; reading from data eeprom
02B3 0A89           01374         incf    EEADR,F         ; adv address pointer for future read
02B4 009E           01375         movwf   BIN4            ; lo byte BIN4 for freq display
02B5 108C           01376         bcf     FLAG,DP         ; clear decimal point flag
02B6 0008           01377         return                  ; finished
                    01378
                    01379 ;*********************************************************************
                    01380 ;* ClrRow1
                    01381 ;* SameAs20
                    01382 ;* This subroutine clears line 1 of LCD and switches off LEDs. Entry
                    01383 ;*  point SameAs20 allows clearing some other number of character
                    01384 ;*  positions starting from line 1 of LCD. Alll LEDs are turned off,
                    01385 ;*  flags for LEDs also Cursor pointer of LCD is restored to first pos
                    01386 ;*  of line 1 at exit of subroutine.
                    01387 ;*
                    01388 ;* Input variables:
                    01389 ;*          entry point ClrRow1:  none
                    01390 ;*          entry point SameAs20: W=number of characters to be cleared
                    01391 ;* Output variables:
                    01392 ;*          CHARCOU is decremented by the number of cleared characters
                    01393 ;*          SHOWCOU is cleared to 0
                    01394 ;*********************************************************************
                    01395 ;* Row1
                    01396 ;*  This subroutine moves cursor to pos 0 of row 1 on LCD.
                    01397 ;*
                    01398 ;*  Input variables: none
                    01399 ;*  Output variables: none
                    01400 ;*********************************************************************
02B7                01401 ClrRow1
02B7 3014           01402         movlw   .20             ; 20 spaces (one row) to write
02B8                01403 SameAs20
02B8 0097           01404         movwf   SHOWCOU         ; SHOWCOU = space counter
02B9 22BF           01405         call    Row1            ; move cursor to row 1
02BA                01406 LoopClD
02BA 23D7           01407         call    Blank           ; print one space
02BB 0B97           01408         decfsz  SHOWCOU,F       ; SHOWCOU = space counter
02BC 2ABA           01409         goto    LoopClD         ; loop if not yet 20 spaces
                    01410
02BD 301F           01411         movlw   1fh             ; bits 7,6,5 (LED flags) reset
02BE 058C           01412         andwf   FLAG,F          ; turn LED flags off
02BF                01413 Row1
02BF 3080           01414         movlw   080h            ; command for line 1
02C0 2BCF           01415         goto    WrComL          ; go write command
                    01416
                    01417 ;*******  SERIAL CODE RECEIVER
                    01418 ;*********************************************************************
                    01419 ;* Mode2Show
```

```
                    01420 ;* This subroutine advances SHOWCOU in range 0...5, and then continues
                    01421 ;*  to subroutine Show2
                    01422 ;*****************************************************************
02C1                01423 Mode2Show
02C1 0A97           01424         incf    SHOWCOU,F       ; advance SHOWCOU
02C2 1917           01425         btfsc   SHOWCOU,2       ; bits 1 & 2 will be set if SHOWCOU...
02C3 1C97           01426         btfss   SHOWCOU,1       ; ...is equal to 5...
02C4 2AC6           01427         goto    Show2           ; ...if not, skip clearing
02C5 0197           01428         clrf    SHOWCOU         ; cycle show counter in range 0...5
                    01429
                    01430 ;*****************************************************************
                    01431 ;* Show2
                    01432 ;* This subroutine prints the 7 bytes of Buffer (+0,+7,+14,+21,+28 or
                    01433 ;*  +35) in hex mode at line 1, and the same bytes in ASCII at line 2.
                    01434 ;*  ASCII representation is with bit 7 reset, and non-printables (<20h)
                    01435 ;*  are printed as dots
                    01436 ;*
                    01437 ;* Input variables:
                    01438 ;*     SHOWCOU, denotes which group of 7 bytes will be displayed
                    01439 ;* Output variables:
                    01440 ;*     CHARCOU is decremented by the number of characters printed
                    01441 ;*****************************************************************
02C6                01442 Show2
02C6 3025           01443         movlw   BUFFER-1        ; source pointer for reading -1
02C7 1917           01444         btfsc   SHOWCOU,2       ; if bit 2 of SHOWCOU set...
02C8 3E1C           01445         addlw   .28             ; ...then add 28 (4 groups) to pointer
02C9 1897           01446         btfsc   SHOWCOU,1       ; if bit 1 of SHOWCOU set...
02CA 3E0E           01447         addlw   .14             ; ...then add 14 (2 groups) to pointer
02CB 1817           01448         btfsc   SHOWCOU,0       ; if bit 0 of SHOWCOU set...
02CC 3E07           01449         addlw   .7              ; ...then add 7 to pointer
02CD 0084           01450         movwf   FSR             ; FSR on (1st byte pos)-1 to display
                    01451
02CE 22BF           01452         call    Row1            ; move cursor to row 1
02CF 3007           01453         movlw   .7              ; bytes to display
02D0 008F           01454         movwf   SCRATCH         ; SCRATCH = byte display counter
02D1                01455 Hex7
02D1 0A84           01456         incf    FSR,F           ; adv pointer (it was x-1 at beginning)
02D2 0E00           01457         swapf   INDF,W          ; move hi nibble to display1. hex digit
02D3 23F8           01458         call    HexDigit        ; display 1st digit in hex mode
02D4 0800           01459         movf    INDF,W          ; move lo nibble to disp 2nd hex digit
02D5 23F8           01460         call    HexDigit        ; display 2nd digit in hex mode
02D6 23D7           01461         call    Blank           ; blank after hex number
                    01462
02D7 0B8F           01463         decfsz  SCRATCH,F       ; SCRATCH = byte counter
02D8 2AD1           01464         goto    Hex7            ; loop if not yet 7 bytes
                    01465
02D9 0804           01466         movf    FSR,W           ; FSR = read pointer
02DA 3EF9           01467         addlw   -.7             ; restore it for ASCII mode printing
02DB 0084           01468         movwf   FSR             ; FSR on (1st byte pos)-1 to display
                    01469
02DC 23CE           01470         call    Row2            ; move cursor to row 2
02DD 3007           01471         movlw   .7              ; bytes to display
02DE 008F           01472         movwf   SCRATCH         ; SCRATCH = byte display counter
02DF                01473 Ascii7
02DF 0A84           01474         incf    FSR,F           ; adv pointer (it was x-1 at beginning)
02E0 0800           01475         movf    INDF,W          ; read byte
02E1 397F           01476         andlw   7fh             ; reduce ascii representation to 7 bits
                    01477
02E2 3EE0           01478         addlw   -20h            ; test if byte < 20h
02E3 3E20           01479         addlw   20h             ; restore previous value
02E4 1803           01480         btfsc   STATUS,C        ; C is set if byte < 20h
02E5 30A5           01481         movlw   0a5h            ; represent non-printables as dots
                    01482                                 ; (0a5h = dot)
02E6 23D8           01483         call    Char            ; display ascii char
                    01484
02E7 0B8F           01485         decfsz  SCRATCH,F       ; SCRATCH = byte counter
```

```
02E8 2ADF          01486         goto    Ascii7          ; loop if not yet 7 bytes
                   01487
02E9 28ED          01488         goto    Farm2           ; go back to user interface
                   01489
                   01490 ;*************************************************************************
                   01491 ;* Mode2Go
                   01492 ;* This is entry point for Start command in mode 2 (serial code rcver)
                   01493 ;*  Line 1 and first 7 positions (ASCII chars) of line 2 on LCD is clr.
                   01494 ;*  Here, buffer is cleared and a sequence of 42 bytes are received and
                   01495 ;*  written to buffer. Manual break (key 2) jumps to Break handling.
                   01496 ;* This protocol is used: High start bit, 7 or 8 data (true) bits, 0 or
                   01497 ;*  1 parity bit (not written to memory) and 1 low stop bit (not tested
                   01498 ;*  for validity). If RXBITS,2 is set then the input is inverted.
                   01499 ;* Baud rates 1200-115200 are supported.
                   01500 ;* Note: no receive errors are detected nor indicated.
                   01501 ;*
                   01502 ;* Input variables:
                   01503 ;*  RXRATE (range 0...7),which affects timing loaded via table BaudRate
                   01504 ;*  RXBITS, bits 0-2 significant:bit0=parity,bit1=7/8 bits,bit2=inverse
                   01505 ;*
                   01506 ;*  Output variables:
                   01507 ;*  Buffer (42 bytes) loaded with bytes received, all unreceived bytes
                   01508 ;*  represented as 00s.
                   01509 ;*************************************************************************
02EA               01510 Mode2Go
02EA 1106          01511         bcf     PORTB,2         ; clear hi-imp probe output...
02EB 190D          01512         btfsc   RXBITS,2        ; ...let it be low if polarity bit clr
02EC 1506          01513         bsf     PORTB,2         ; set hi-imp output if polarity bit set
02ED 23B7          01514         call    ClrBuf          ; clear wholw buffer
02EE 302F          01515         movlw   .47             ; .47 blanks to clear displayed values
02EF 22B8          01516         call    SameAs20        ; clear displayed HEX and ASCII values
                   01517
02F0 3026          01518         movlw   BUFFER          ; start of buffer...
02F1 0084          01519         movwf   FSR             ; ...assigned to destination pointer
                   01520
02F2 081D          01521         movf    RXRATE,W        ; RXRATE = selected rate in range 0...7
02F3 3E68          01522         addlw   BaudRate        ; add to timing constant table offset
02F4 23B6          01523         call    PclSub          ; get rate to W
02F5               01524 RX42Bytes
02F5 008E          01525         movwf   DJNZ            ; baudrate timing factor to time cnter
                   01526
02F6 0194          01527         clrf    COUNT           ; this is to preset bit counter to 8...
02F7 1594          01528         bsf     COUNT,3         ; ...and not to disturb W
                   01529
02F8 1C0D          01530         btfss   RXBITS,0        ; RXBITS,0 is set if 7 bits selected
02F9 1C8D          01531         btfss   RXBITS,1        ; RXBITS,1 is set if parity bit select
02FA 0A94          01532         incf    COUNT,F         ; if not(RXBITS and 3 = 2) then COUNT=9
                   01533
02FB 180D          01534         btfsc   RXBITS,0        ; RXBITS,0 is set if 7 bits selected
02FC 188D          01535         btfsc   RXBITS,1        ; RXBITS,1 is set if parity bit select
02FD 0394          01536         decf    COUNT,F         ; ifnot(RXBITS and 3=1)reduce to 8 or 7
                   01537
02FE 1D0D          01538         btfss   RXBITS,2        ; RXBITS,2 set if inverse polar slctd
02FF 2B07          01539         goto    GetSp2          ; jump to true polarity if RXBITS,2 clr
                   01540
0300               01541 GetSp1                          ; ----- inverse rx
0300 1E05          01542         btfss   PORTA,4         ; test input status...
0301 2B00          01543         goto    GetSp1          ; ...and loop if still low
0302               01544 GetStart1
0302 1D85          01545         btfss   PORTA,3         ; test status of key 2...
0303 28E8          01546         goto    BrkRS           ; ...and jump to Break if presseed
0304 1A05          01547         btfsc   PORTA,4         ; test input status...
0305 2B02          01548         goto    GetStart1       ; ...and loop if still high
0306 2B0D          01549         goto    StartFound      ; falling edge detected: start recept
                   01550
0307               01551 GetSp2                          ; ----- true rx
```

```
0307 1A05          01552         btfsc   PORTA,4         ; test input status...
0308 2B07          01553         goto    GetSp2          ; ...and loop if still high
0309               01554 GetStart2
0309 1D85          01555         btfss   PORTA,3         ; test status of key 2...
030A 28E8          01556         goto    BrkRS           ; ...and jump to Break if presseed
030B 1E05          01557         btfss   PORTA,4         ; test input status...
030C 2B09          01558         goto    GetStart2       ; ...and loop if still low
                   01559                                 ; rising edge detected: start reception
030D               01560 StartFound
                   01561                                 ; 2-9 t from starting edge
030D               01562 HalfBit
030D 1686          01563         bsf     PORTB,5         ; 1*W led P on
030E 0000          01564         nop                     ; 1*W waist one cycle
030F 0B8E          01565         decfsz  DJNZ,F          ; 1*W DJNZ = timing constant counter
0310 2B0D          01566         goto    HalfBit         ; 2*W loop if not half bit timing passd
                   01567
0311               01568 RX8Bits
0311 008E          01569         movwf   DJNZ            ;  1     move timing constant to cnter
0312               01570 OneBit
0312 23A0          01571         call    Waist8T         ; 8 8 * W  waist 8 t
0313 0B8E          01572         decfsz  DJNZ,F          ; 1 2 * W  DJNZ=timing constant counter
0314 2B12          01573         goto    OneBit          ; 2 - * W  loop if not 1 bit time passd
                   01574
0315 23A2          01575         call    Waist6T         ;  6        waist 6 t
                   01576
0316 0C85          01577         rrf     PORTA,F         ;  1   <--- move input status to C
0317 0C80          01578         rrf     INDF,F          ;  1        place C in input rotor
                   01579
0318 0B94          01580         decfsz  COUNT,F         ;  1        COUNT = bit counter
0319 2B11          01581         goto    RX8Bits         ;  2 total 22t + (w-1) * 11
                   01582                                 ;------------------received byte @ INDF
031A 180D          01583         btfsc   RXBITS,0        ; test if parity bit selected...
031B 0D80          01584         rlf     INDF,F          ; ...and discard parity bit if received
                   01585
031C 1D0D          01586         btfss   RXBITS,2        ; test if inverse polarity selected...
031D 0980          01587         comf    INDF,F          ; ...and complement if true parity
                   01588
031E 1003          01589         bcf     STATUS,C        ; clear 8th bit for 7-bit mode
031F 188D          01590         btfsc   RXBITS,1        ; test if 7-bit mode selected and...
0320 0C80          01591         rrf     INDF,F          ; ..rotate 8th(zero) bit if 7bits slctd
                   01592
0321 0A84          01593         incf    FSR,F           ; advance destination pointer
0322 1B04          01594         btfsc   FSR,6           ; bits 4 and 6 will both be set if...
0323 1E04          01595         btfss   FSR,4           ; ... end of buffer+1 reached
0324 2AF5          01596         goto    RX42Bytes       ; loop if not yet FSR=50h
0325 2AC6          01597         goto    Show2           ; over: go show received bytes
                   01598
                   01599 ;*******  FREQUENCY COUNTER
                   01600 ;*********************************************************************
                   01601 ;* GoPresc
                   01602 ;* Prescaler factor (variable PRESC, in range 0...3) is advanced
                   01603 ;*  (executes when key 2 is pressed in frequency counter mode)
                   01604 ;*********************************************************************
0326               01605 GoPresc
0326 0A9A          01606         incf    PRESC,F         ; advance prescaler
0327 111A          01607         bcf     PRESC,2         ; and cycle prescaler in range 0...3
                   01608
                   01609 ;*********************************************************************
                   01610 ;* FreqEp
                   01611 ;* Frequency counter entry point.
                   01612 ;* Subroutine WrParam does this: Displays message "Frequency" in row 1,
                   01613 ;*  counter range (taken from table RangeTab) & resolution (taken from
                   01614 ;*  table PrescTab) in row2.
                   01615 ;* LEDs are turned OFF, and the main counter (BIN4, 4 bytes) cleared.
                   01616 ;*  The following is used to count pulses:
                   01617 ;* State of TMR0 is written to BIN4+0 and sequencialy tested, and when
```

```
                     01618 ;*  bit7 of current value detected as 0 and the previous one was 1, the
                     01619 ;*  overflow is considered. In that case, state of BIN4+1 is advanced,
                     01620 ;*  extended to BIN4+2. After 500ms,the 32-bit value of BIN4 is shifted
                     01621 ;*  left in a total of PRESC+2 times, to get multiply by 4,8,16 or
                     01622 ;*  32. Then BIN+4 (4 bytes) is converted to ASCII and printed on LCD.
                     01623 ;* This routine does not call keyboard routine, as accurate timing of
                     01624 ;*  500 ms must be generated for counting (high count register is
                     01625 ;*  CHARCOU lo count SHOWCOU). Instead of this, there is the individual
                     01626 ;*  routine for key 1 and key 2 test, and also the countdown timer for
                     01627 ;*  automatic power-off (registers TIMOUTL, TIMOUTH).
                     01628 ;* If key 1 is  pressed, mode 1 (analyzer) is entered. If key 2 is
                     01629 ;*  pressed, prescaler value is advanced.
                     01630 ;*
                     01631 ;* Note: code from label "Loop500A" to comment "; 500 ms timeout" is
                     01632 ;*  real time code. If the # of cycles is changed, then the literals
                     01633 ;*  0f4h+1 and 24h+1 written to CHARCOU and SHOWCOU must be readjusted.
                     01634 ;*
                     01635 ;* Input variables: PRESC (affects prescaler factor)
                     01636 ;* Output variables: none
                     01637 ;*********************************************************************
0328                 01638 FreqEp                          ; mode 3: frequency counter
0328 2364            01639         call    WrParam         ; print "Frequency" and "xxMHz/Rxx"
0329 21BF            01640         call    KeysOff         ; test both keys off for 34 ms and
                     01641                                 ; initialize 8 min auto off sequence
032A                 01642 Count500
032A 30D3            01643         movlw   0d2h+REL        ; right arrow position
032B 23CF            01644         call    WrComL          ; move cursor on right arrow
                     01645
032C 019F            01646         clrf    BIN4+1          ; clear next counter byte
032D 01A0            01647         clrf    BIN4+2          ; clear next counter byte
032E 01A1            01648         clrf    BIN4+3          ; clear next counter byte
032F 018F            01649         clrf    SCRATCH         ; initialize TMR0 overflow detector
                     01650
0330 30F5            01651         movlw   0f4h+1          ; high loop counter for 500 ms
0331 0096            01652         movwf   CHARCOU         ; CHARCOU = hi byte counter
0332 3025            01653         movlw   24h+1           ; 0f424h=.62500 cycles=1250000,T=500 ms
0333 0097            01654         movwf   SHOWCOU         ; SHOWCOU = lo byte counter
                     01655
0334 081A            01656         movf    PRESC,W         ; PRESC = prescaler factor selected
0335 3E20            01657         addlw   20h             ; for PRESC 0,1,2,3  w=20h,21h,22h,23h
0336 2278            01658         call    ToOption        ; here is clrf TMR0 also
0337                 01659 Loop500A
0337 1D85            01660         btfss   PORTA,3         ; 2   test key 2 status and
0338 2B26            01661         goto    GoPresc         ; -   jump to "prescaler change" if hit
0339                 01662 Loop500
0339 0801            01663         movf    TMR0,W          ; 1 1 1   only place where TMR0 is read
033A 009E            01664         movwf   BIN4            ; 1 1 1   rtcc ---> freq0
                     01665
033B 0D1E            01666         rlf     BIN4,W          ; 1 1 1   carry <--- TMR0.7
033C 0D8F            01667         rlf     SCRATCH,F       ; 1 1 1   rotor <--- carry
                     01668
033D 080F            01669         movf    SCRATCH,W       ; 1 1 1   SCRATCH=TMR0 overflow detect
033E 3903            01670         andlw   3               ; 1 1 1   mask 2 LSbs for edge detect
033F 3A02            01671         xorlw   2               ; 1 1 1   00000000 if 1 <--- 0
                     01672
0340 1D03            01673         btfss   STATUS,Z        ; 1 2 2 |   skip if TMR0 overflow
0341 2B43            01674         goto    NotOvf1         ; 2 - - |   if nz
                     01675                                 ;       | 5T any case
0342 0A9F            01676         incf    BIN4+1,F        ; - 1 1 |   nsb
0343 1903            01677 NotOvf1 btfsc   STATUS,Z        ; 2 2 1 |   skip MSB adv ifnot overflow
0344 0AA0            01678         incf    BIN4+2,F        ; - - 1 |   msb
                     01679
0345 309B            01680         movlw   Head4-1         ; 1   initialize "Battery" message
0346 1D05            01681         btfss   PORTA,2         ; 2   test key 1 status and...
0347 2933            01682         goto    Mode4           ; -   got shortcut to mode 4 if pressed
                     01683
```

```
0348 0B97          01684         decfsz  SHOWCOU,F       ; 1 (2)   lo loop counter
0349 2B37          01685         goto    Loop500A        ; 2 (-)   20T total
                   01686
034A 0B96          01687         decfsz  CHARCOU,F       ;   (1)   hi loop counter
034B 2B39          01688         goto    Loop500         ;   (2)   20T total
                   01689
                   01690                                 ; --- 500 ms timeout here
034C 0A1A          01691         incf    PRESC,W         ; prepare for x2 multiply: PRESC incr
034D 0096          01692         movwf   CHARCOU         ; CHARCOU = multiply factor counter
034E 0A96          01693         incf    CHARCOU,F       ; and prescaler constant incr again
034F               01694 ShLoop                          ; BIN4 = BIN4 * 2 total (PRESC+2) times
034F 1003          01695         bcf     STATUS,C        ; clear C to allow clean x2 multiply
0350 0D9E          01696         rlf     BIN4, F         ; low byte x2 multiply
0351 0D9F          01697         rlf     BIN4+1,F        ; next byte x2 multiply
0352 0DA0          01698         rlf     BIN4+2,F        ; next byte x2 multiply
0353 0DA1          01699         rlf     BIN4+3,F        ; highest byte x2 multiply
                   01700
0354 0B96          01701         decfsz  CHARCOU,F       ; CHARCOU = multiply factor counter
0355 2B4F          01702         goto    ShLoop          ; loop if not PRESC*2 times multiplied
                   01703
0356 308A          01704         movlw   8ah             ; frequency display position
0357 23CF          01705         call    WrComL          ; cursor to freq display position
0358 238C          01706         call    Print8          ; print the frequency in 8-digit ASCII
                   01707
0359 0B9B          01708         decfsz  TIMOUTL,F       ; TIMOUTL=lo byte auto power off cnter
035A 2B2A          01709         goto    Count500        ; inner loop
035B 151B          01710         bsf     TIMOUTL,2       ; TIMOUTL is init to 4 passes instead
                   01711                                 ; of 256, 4*256*500ms=512s=8.5min appr
                   01712
035C 0B9C          01713         decfsz  TIMOUTH,F       ; TIMOUTH=hi byte auto power off cnter
035D 2B2A          01714         goto    Count500        ; loop if not yet 8.5 min
                   01715
035E 29BC          01716         goto    Suicide         ; 8.5 min timeout - go switch power off
                   01717
                   01718 ;*******  BINARY TO ASCII CONVERSION
                   01719 ;*****************************************************************
                   01720 ;* Headline
                   01721 ;* Clears SHOWCOU (Headline2 skips this), Clears LCD, prints right
                   01722 ;*  arrow @ last pos of row2, prints message addresed by W+1 at page 0.
                   01723 ;* Terminator is last character with bit 7 set.
                   01724 ;*
                   01725 ;* Input variables: W+1 addresses string (on page 0) to be printed
                   01726 ;* Output variables:
                   01727 ;*    CHARCOU is decremented by the number of characters printed
                   01728 ;*****************************************************************
035F               01729 Headline
035F 0197          01730         clrf    SHOWCOU         ; initialize show group counter
0360               01731 Headline2
0360 008F          01732         movwf   SCRATCH         ; move input parameter to SCARTCH
0361 303B          01733         movlw   .59             ; .59 characters to clear
0362 22B8          01734         call    SameAs20        ; clear all but right arrow
0363 2BDE          01735         goto    GoWrite         ; print headline message on LCD
                   01736
                   01737 ;*****************************************************************
                   01738 ;* WrParam
                   01739 ;* Prints message "Frequency" in line 1 and parameters for frequency
                   01740 ;*  counter in line 2.
                   01741 ;* Text XXMHz/RYY is printed, where XX is taken from RangeTab, and YY
                   01742 ;*  from PrescTab.
                   01743 ;*
                   01744 ;* Input variables:
                   01745 ;*    PRESC, affects displayed frequency range and resolution
                   01746 ;* Output variables:
                   01747 ;*    CHARCOU is decremented by the number of characters printed
                   01748 ;*****************************************************************
                   01749 ;* Print255
```

```
                     01750 ;* Entry point Print255 converts 8-bit binary value (<100) in BIN4 to
                     01751 ;*  2-digit ASCII and prints it on LCD, without decimal point. Leading
                     01752 ;* zeros are printed.
                     01753 ;*
                     01754 ;* Input variables: W, binary number (0-99) to be converted and printed
                     01755 ;* Output variables:
                     01756 ;*    CHARCOU is decremented by the number of characters printed
                     01757 ;*********************************************************************
                     01758 ;* Print3
                     01759 ;* Entrypoint Print3 converts 16-bit binary value (<1000) in BIN4 to 2-
                     01760 ;*  or 3-digit ASCII and prints it on LCD. Leading zero is skipped only
                     01761 ;*  if value is <100. Decimal point is printed between tens and ones if
                     01762 ;*  FLAG,DP is set, otherwise decimal point is omited.
                     01763 ;*
                     01764 ;* Input variables: BIN4 (2 bytes, LSB first, in range 0-999) binary
                     01765 ;*                       number to be converted and printed
                     01766 ;* Output variables:
                     01767 ;*    CHARCOU is decremented by the number of characters printed
                     01768 ;*********************************************************************
                     01769 ;* PrintBR
                     01770 ;* Entry point PrintBR does the same as PRINT3, but the low byte value
                     01771 ;*  is in W instead in BIN4+0. This is used for baud rate display.
                     01772 ;*
                     01773 ;* Input variables: W, BIN+1 (2 bytes, LSB in W, MSB in BIN+1, in range
                     01774 ;*  0-999) binary number to be converted and printed
                     01775 ;* Output variables:
                     01776 ;*    CHARCOU is decremented by the number of characters printed
                     01777 ;*********************************************************************
0364                 01778 WrParam                          ; print xxMHz/Rxx
0364 3092            01779         movlw   Head3-1          ; address of message "Frequency"
0365 2360            01780         call    Headline2        ; print message
0366 23CE            01781         call    Row2             ; move cursor to row 2
                     01782
0367 3064            01783         movlw   RangeTab         ; offset of max frequency range table
0368 236E            01784         call    Presc255         ; print max frequency range
                     01785
0369 306F            01786         movlw   TxtHz-1          ; address of message "MHz/"
036A 23DD            01787         call    Write            ; print message
036B 3052            01788         movlw   'R'              ; "R" stands for "Resolution"
036C 23D8            01789         call    Char             ; print "R"
                     01790
036D 3060            01791         movlw   PrescTab         ; offset of resolution table
036E                 01792 Presc255
036E 071A            01793         addwf   PRESC,W          ; add PRESC to offset
036F 23B6            01794         call    PclSub           ; read value from table
0370                 01795 Print255
0370 108C            01796         bcf     FLAG,DP          ; clear decimal point enable flag
0371 019F            01797         clrf    BIN4+1           ; clear hi byte (allow range 00-99)
0372                 01798 PrintBR
0372 009E            01799         movwf   BIN4             ; allow W as input parameter
0373                 01800 Print3                           ; convert BIN4(16),print 3 decimal dgts
0373 01A1            01801         clrf    BIN4+3           ; clear hi byte
0374 01A0            01802         clrf    BIN4+2           ; clear next byte
0375 01A5            01803         clrf    CMP4+3           ; clear hi byte of temporary register
0376 01A4            01804         clrf    CMP4+2           ; clear next byte of temporary register
0377 01A3            01805         clrf    CMP4+1           ; clear next byte of temporary register
                     01806
0378 3064            01807         movlw   .100             ; first digit constant
0379 2383            01808         call    Times            ; # of times goes in BIN4+1 and BIN4?
037A 1D03            01809         btfss   STATUS,Z         ; skip printing if it goes zero times
037B 23C9            01810         call    Num              ; print digit (hundreds) if W>0
                     01811
037C 300A            01812         movlw   .10              ; second digit constant
037D 2383            01813         call    Times            ; how many times it goes in BIN4?
037E 23C9            01814         call    Num              ; print digit (tens)
                     01815
```

```
037F 302E           01816         movlw   '.'             ; decimal point
0380 188C           01817         btfsc   FLAG,DP         ; test decimal pnt flag, skip if reset
0381 23D8           01818         call    Char            ; print decimal point if DP set
0382 2B9B           01819         goto    NumBin4         ; print last digit (ones)
                    01820
                    01821 ;*****************************************************************
                    01822 ;* Times
                    01823 ;* Counts # of times CMP4 (32-bit value) "goes" in BIN4 (32-bit value).
                    01824 ;* BIN4 is sequentialy subtracted by CMP4 and counter COUNT advanced.
                    01825 ;*  When borrow is detected, BIN4 is restored to the last positiv value
                    01826 ;*  (by ADDing CMP4 again), COUNTer decremented and written to W.
                    01827 ;*
                    01828 ;* Input variables: CMP4 (32-bit value), BIN4 (32-bit value)
                    01829 ;* Output variables:
                    01830 ;*    BIN4 (32-bit value) modified to mod(CMP4)
                    01831 ;*    W (in range 0...9) = BIN4 (32-bit value) / CMP4 (32-bit value)
                    01832 ;*****************************************************************
0383                01833 Times
0383 00A2           01834         movwf   CMP4            ; place input param in CMP4 to compare
0384 0194           01835         clrf    COUNT           ; clear result counter
0385                01836 GoTD
0385 0A94           01837         incf    COUNT,F         ; advance result counter
0386 239D           01838         call    Sub4            ; BIN4=BIN4-CMP4  nc if result <0
0387 1803           01839         btfsc   STATUS,C        ; test did it "go"?
0388 2B85           01840         goto    GoTD            ; loop if so
0389 23A5           01841         call    Add4            ; BIN4=BIN4+CMP4  c set if ovf
038A 0314           01842         decf    COUNT,W         ; W=# of times CMP4 goes in BIN4(32bit)
038B 0008           01843         return                  ; result in W
                    01844
                    01845 ;*****************************************************************
                    01846 ;* Print8
                    01847 ;* This subroutine converts 32-bit value in BIN4 (low byte first), to
                    01848 ;*  8-dig ASCII and prints to LCD. Leading zeros are printed as blanks.
                    01849 ;* Table DecTab (21 words, must be at page 0 if PCLATH=0) used in conv.
                    01850 ;*
                    01851 ;* Input variables: BIN4 (32-bit value, <.100,000,000)
                    01852 ;* Output variables:
                    01853 ;*    CHARCOU is decremented by the number of characters printed
                    01854 ;*****************************************************************
038C                01855 Print8                          ; bin2dec conv BIN4(32), print 8 digits
038C 3033           01856         movlw   DecTab-1        ; inici tab ptr
038D 008F           01857         movwf   SCRATCH         ; SCRATCH = tab ptr
038E 118C           01858         bcf     FLAG,RIPPLE     ; zeros initialy print as blanks,until
                    01859                                 ; ...first non-zero appears
038F                01860 Cif7
038F 01A5           01861         clrf    CMP4+3          ; clear CMP4+3, it is =0 in all cases
0390 23B4           01862         call    PclSub2         ; get constant from table
0391 00A4           01863         movwf   CMP4+2          ; load dec. const from table in CMP4+2
0392 23B4           01864         call    PclSub2         ; get constant from table
0393 00A3           01865         movwf   CMP4+1          ; load dec. const from table in CMP4+1
0394 23B4           01866         call    PclSub2         ; get constant from table
0395 2383           01867         call    Times           ; how many times CMP4 goes in BIN4?
0396 23C3           01868         call    NZNum           ; print if w>0 or RIPPLE=1, else blank
                    01869
0397 080F           01870         movf    SCRATCH,W       ; SCRATCH = table pointer
0398 3EB9           01871         addlw   .237-DecTab     ; test if end of table
0399 1C03           01872         btfss   STATUS,C        ; C set if end of table
039A 2B8F           01873         goto    Cif7            ; if not end of table loop (will be 7x)
039B                01874 NumBin4
039B 081E           01875         movf    BIN4,W          ; last digit is in BIN4
039C 2BC9           01876         goto    Num             ; last digit must be printed always
                    01877
                    01878 ;*****************************************************************
                    01879 ;* Sub4
                    01880 ;* Subtract CMP4 (32-byte value) from 32-bit value in BIN4, lo byte 1st
                    01881 ;* This is performed as adding of negative value of CMP4. Negating is
```



```
                    01882 ;*  performed as comlementing and incrementing by 1.
                    01883 ;* Note: Incrementing by 1 is performed on least significant byte only,
                    01884 ;*  without 32-bit extension,for code space saving. This will not cause
                    01885 ;*  error in this case, as the number of all possible values for CMP4+0
                    01886 ;*  is limited and none of them is equal to 0FFh before incrementing
                    01887 ;*  (all possible values are taken from table DecTab, & are: 0ah, 64h,
                    01888 ;*  0e8h, 10h, 0a0h, 40h and 80h, and their negative values).
                    01889 ;*  However, this is valid if this subroutine is used for decimal
                    01890 ;*  conversion only, and if it is used for some other application,
                    01891 ;*  extension to 32-bit should be added after incrementing.
                    01892 ;*
                    01893 ;* Input variables: BIN4 (32-bit value), CMP4 (32-bit value)
                    01894 ;* Output variables:
                    01895 ;*    BIN4 (32-bit value)
                    01896 ;*    STATUS,C denotes the sign of result: if cleared, output value
                    01897 ;*              is negative (there is borrow)
                    01898 ;*
                    01899 ;*  Note: Entry points Waist8T and Waist6T are used only by some
                    01900 ;*  real-time routines, in that case the instructions are dummy
                    01901 ;*********************************************************************
039D                01902 Sub4                            ; 32-bit sub: BIN4 = BIN4 - CMP4
                    01903                                 ; NC if result<0
039D 239F           01904         call    NegCmp          ; negate CMP (32 bits) first
039E 23A5           01905         call    Add4            ; add as negative value
039F                01906 NegCmp
039F 09A2           01907         comf    CMP4+0,F        ; complement low byte
03A0                01908 Waist8T
03A0 09A3           01909         comf    CMP4+1,F        ; complement next byte
03A1 09A4           01910         comf    CMP4+2,F        ; complement next byte
03A2                01911 Waist6T
03A2 09A5           01912         comf    CMP4+3,F        ; complement high byte
03A3 0AA2           01913         incf    CMP4+0,F        ; neg = complement + 1
                    01914                                 ; (no need test overflow here,it will
                    01915                                 ; ...never reach 0 after incrementing)
03A4 0008           01916         return                  ; finished
                    01917
                    01918 ;*********************************************************************
                    01919 ;* Add4
                    01920 ;* Add CMP4 (32-byte value) to BIN4 (32-byte value). 4-instr. groups
                    01921 ;* (movf-btfsc-incfsz-addwf) used instead of non-existing ADD W/ CARRY.
                    01922 ;* Input variables: BIN4 (32-bit value), CMP4 (32-bit value)
                    01923 ;* Output variables: BIN4 (32-bit value), 33th bit in STATUS,C
                    01924 ;*********************************************************************
03A5                01925 Add4                            ; 32-bit add: BIN4 = BIN4 + CMP4
03A5 0822           01926         movf    CMP4,W          ; low byte
03A6 079E           01927         addwf   BIN4,F          ; low byte add
                    01928
03A7 0823           01929         movf    CMP4+1,W        ; next byte
03A8 1803           01930         btfsc   STATUS,C        ; skip to simple add if C was reset
03A9 0F23           01931         incfsz  CMP4+1,W        ; add C if it was set
03AA 079F           01932         addwf   BIN4+1,F        ; next byte add if NZ
                    01933
03AB 0824           01934         movf    CMP4+2,W        ; next byte
03AC 1803           01935         btfsc   STATUS,C        ; skip to simple add if C was reset
03AD 0F24           01936         incfsz  CMP4+2,W        ; add C if it was set
03AE 07A0           01937         addwf   BIN4+2,F        ; next byte add if NZ
                    01938
03AF 0825           01939         movf    CMP4+3,W        ; high byte
03B0 1803           01940         btfsc   STATUS,C        ; skip to simple add if C was reset
03B1 0F25           01941         incfsz  CMP4+3,W        ; add C if it was set
03B2 07A1           01942         addwf   BIN4+3,F        ; high byte add if NZ
                    01943
03B3 0008           01944         return                  ; finished
                    01945
                    01946 ;*********************************************************************
                    01947 ;* PclSub is used for indirect addressing
```

```
                    01948 ;* PclSub1 uses SCRATCH instead of W as input parameter
                    01949 ;* PclSub2 advances pointer SCRATCH before executong
                    01950 ;*
                    01951 ;* Note: PCLATH=0 in all cases. So all tables pointed by this routine
                    01952 ;*  are on page 0
                    01953 ;*****************************************************************
03B4                01954 PclSub2
03B4 0A8F           01955         incf    SCRATCH,F       ; advance table pointer
03B5                01956 PclSub1
03B5 080F           01957         movf    SCRATCH,W       ; move table pointer to W
03B6                01958 PclSub
03B6 0082           01959         movwf   PCL             ; jump to address pointed by PCLATH,W
                    01960
                    01961 ;*****************************************************************
                    01962 ;* ClrBuf
                    01963 ;* ClrRam
                    01964 ;* Subroutine ClrBuf clears BUFFER (42 bytes)
                    01965 ;* Entry point ClrRam allows some other start point for claring. It
                    01966 ;*  clears internal RAM from address in W to the location 7Fh.
                    01967 ;*  (locations 50h-7Fh, which do not exist in 16F84, are dummy).
                    01968 ;*
                    01969 ;* Both entry points continue to disabling Enable signal for LCD and
                    01970 ;*  33.8 ms timing loop
                    01971 ;*
                    01972 ;* Input variables:
                    01973 ;*  W is the start addr if area to be cleared (ClrRam entry point only)
                    01974 ;* Output variables: none
                    01975 ;*****************************************************************
03B7                01976 ClrBuf
03B7 3026           01977         movlw   BUFFER          ; get start address of buffer
03B8                01978 ClrRam
03B8 0084           01979         movwf   FSR             ; FSR = dest pointer for clearing
03B9                01980 Zeros
03B9 0180           01981         clrf    INDF            ; clear one byte
03BA 0A84           01982         incf    FSR,F           ; advance dest pointer
03BB 1F84           01983         btfss   FSR,7           ; test if end of RAM...
03BC 2BB9           01984         goto    Zeros           ; ...if not, loop - else move LEDs
                    01985
                    01986 ;*****************************************************************
                    01987 ;*  Entry point DisEna30: Remove enable and discharge signal, and
                    01988 ;*                          refresh LEDs. Then loop 33.8 ms
                    01989 ;*  Entry point Wait30: Loop 33.8 ms
                    01990 ;*  Input variables: none
                    01991 ;*  Output variables: none
                    01992 ;*****************************************************************
03BD                01993 DisEna30
03BD 23EF           01994         call    DisEna          ; disable discharging output signal
03BE                01995 Wait30
03BE 0194           01996         clrf    COUNT           ; COUNT=time loop cnter,to wait 33.8ms
03BF                01997 GoWait30
03BF 23D0           01998         call    loop130         ; waist 130 us
03C0 0B94           01999         decfsz  COUNT,F         ; COUNT = time loop counter
03C1 2BBF           02000         goto    GoWait30        ; loop if not yet 256 passes
03C2 0008           02001         return                  ; timing over
                    02002
                    02003 ;*****************************************************************
                    02004 ;* NZNum
                    02005 ;* Num
                    02006 ;* Print numeric value in W (in range 0...9) on LCD. If FLAG,RIPPLE is
                    02007 ;*  cleared, 0 is printed as blank. If non-zero numeric is printed, it
                    02008 ;*  automatically sets FLAG,RIPPLE.
                    02009 ;* 0 (30h) prints as capital O (4Fh), for improved readibility, as 0
                    02010 ;*  may easily be substituted by 8 on LCD. This changing 0 to O is not
                    02011 ;*  performed only in ASCII representation of recorded bytes in serial
                    02012 ;*  code receiver.
                    02013 ;* Entry point NUM prints numeric unconditionally, undependently of bit
```

```
                    02014 ;*  FLAG,RIPPLE.
                    02015 ;*
                    02016 ;* Input variables:
                    02017 ;*    W (0...9), number to be printed at current cursor position of LCD
                    02018 ;* Output variables:
                    02019 ;*    CHARCOU is decremented by the number of characters printed
                    02020 ;************************************************************************
03C3                02021 NZNum                           ; same aso Num, only blank instead of 0
03C3 198C           02022         btfsc   FLAG,RIPPLE     ; test if RIPPLE bit set...
03C4 2BC9           02023         goto    Num             ; ...if RIPPLE set, no more blanks
03C5 3E00           02024         addlw   0               ; set Z flag if W=0
03C6 1903           02025         btfsc   STATUS,Z        ; is it = 0 ?
03C7 2BD7           02026         goto    Blank           ; if so, jump to space routine
03C8 158C           02027         bsf     FLAG,RIPPLE     ; if>0,clr RIPPLE bit, no more blanks
03C9                02028 Num
03C9 390F           02029         andlw   0fh             ; isolate low nibble
03CA 1903           02030         btfsc   STATUS,Z        ; is it = 0 ?
03CB 301F           02031         movlw   'O'-30h         ; if so, initialize capital O
03CC 3E30           02032         addlw   30h             ; adjust ASCII for numeric
03CD 2BD8           02033         goto    Char            ; print digit
                    02034
                    02035 ;*******  LCD ROUTINES
                    02036 ;************************************************************************
                    02037 ;*  All these entry ponts of this subroutine are used in program:
                    02038 ;*
                    02039 ;*  Row2:    Issues command to move cursor to row2 of LCD,and loops 130
                    02040 ;*             us,to allow time for LCD controller to execute command
                    02041 ;*  WrComl:  Issues command in W to the LCD, and loops 130 us, to allow
                    02042 ;*             time to LCD controller to execute the command
                    02043 ;*  loop130: Loops 130 us including call and return
                    02044 ;*  GoLoop:  Loops W*2 us
                    02045 ;*  Loop7:   Same as GoLoop, only 2t shorter (for smpl rate routine)
                    02046 ;************************************************************************
03CE                02047 Row2
03CE 30C0           02048         movlw   0c0h            ; command for line 2
                    02049
03CF                02050 WrComL                          ; issues command in W
03CF 23E5           02051         call    WrCom           ; write command in LCD
03D0                02052 loop130                         ; * waist 130 us
03D0 018E           02053         clrf    DJNZ            ; this is to init DJNZ to 40h and...
03D1 170E           02054         bsf     DJNZ,6          ; not to disturb W
03D2                02055 GoLoop
03D2 2BD3           02056         goto    $+1             ; 2    waist 2 t
03D3                02057 Loop7
03D3 0B8E           02058         decfsz  DJNZ,F          ; 1    DJNZ = timing loop counter
03D4 2BD2           02059         goto    GoLoop          ; 2    64x5=320t (128 us)
03D5 0008           02060         return                  ; 2    finished
                    02061
                    02062 ;************************************************************************
                    02063 ;*  All these entry ponts of this subroutine are used in program:
                    02064 ;*
                    02065 ;*  CharBl:  print character in W, blank on LCD and decrement CHARCOU
                    02066 ;*  Blank:   print blank (32h) on LCD and decrement CHARCOU
                    02067 ;*  Char:    print character in W on LCD and decrement CHARCOU
                    02068 ;*  CharNCC: print character in W on LCD without affecting CHARCOU
                    02069 ;*
                    02070 ;*  Note: CHARCOU is used to print fixed format message on LCD, as the
                    02071 ;*  calling routine will add N-CHARCOU blanks to fill area N chars long
                    02072 ;*
                    02073 ;*  Input variables:
                    02074 ;*         all entry points except Blank: W = character to be printed
                    02075 ;*  Output variables:
                    02076 ;*         all entry points except CharNCC: CHARCOU is decremented by 1
                    02077 ;************************************************************************
03D6                02078 CharBl                          ; print char, then blank
03D6 23D8           02079         call    Char            ; print char first
```

```
03D7                02080 Blank                           ; print blank
03D7 3020           02081         movlw   ' '             ; print blank
03D8                02082 Char                            ; print W
03D8 0396           02083         decf    CHARCOU,F       ; decrement character counter
03D9                02084 CharNCC                         ; print W without affecting CHARCOU
03D9 1486           02085         bsf     PORTB,1         ; pull RS hi (data register select)
03DA 2BE6           02086         goto    Skr1            ; continue 4-bit mode writing to LCD
                    02087
                    02088 ;*****************************************************************
                    02089 ;* PrintBrk
                    02090 ;*  Print message "Break" in row 1, pos 0
                    02091 ;*
                    02092 ;* Input variables: none
                    02093 ;* Output variables:
                    02094 ;*    CHARCOU is decremented by the number of characters printed
                    02095 ;*****************************************************************
                    02096 ;* Write
                    02097 ;* Print message addressed by W+1 in line 1
                    02098 ;* Note: Terminator is last character in string with bit 7 set
                    02099 ;*
                    02100 ;* Input variables: W points to string (RETLWs) decremented by 1
                    02101 ;* Output variables:
                    02102 ;*     CHARCOU is decremented by the number of characters printed
                    02103 ;*****************************************************************
03DB                02104 PrintBrk
03DB 22BF           02105         call    Row1            ; move cursor to row 1
03DC 30A2           02106         movlw   BrkMes-1        ; message "Break" address-1
                    02107
03DD                02108 Write                           ; write string addressed by W, end w/ 0
03DD 008F           02109         movwf   SCRATCH         ; SCRATCH = source pointer
03DE                02110 GoWrite                         ; write string at (SCRATCH+1), wnd w/ 0
03DE 23B4           02111         call    PclSub2         ; advance pointer and read pointed byte
03DF 3E80           02112         addlw   80h             ; this is to test if bit 7 was set...
03E0 1803           02113         btfsc   STATUS,C        ; ...if so, C will be set
03E1 2BD8           02114         goto    Char            ; last character was with bit 7 set
03E2 397F           02115         andlw   7fh             ; restore initial character value
03E3 23D8           02116         call    Char            ; print one character
03E4 2BDE           02117         goto    GoWrite         ; loop
                    02118
                    02119 ;*****************************************************************
                    02120 ;* Wrcom
                    02121 ;*  Write command in W to LCD, then loop 130 us
                    02122 ;* Skr1
                    02123 ;*  Allows data write to LCD, if PORTB,1 is set previously
                    02124 ;* Nibble
                    02125 ;*  Write one nibble (W,0-3) to LCD data bus
                    02126 ;*
                    02127 ;* Note: all entry ponits are terminated by 130us timing loop, to allow
                    02128 ;*  LCD controller to execute accepted command/data.
                    02129 ;*
                    02130 ;* Input variables: command in W
                    02131 ;* Output variables: none
                    02132 ;*****************************************************************
03E5                02133 WrCom
03E5 1086           02134         bcf     PORTB,1         ; rs lo (command)
03E6                02135 Skr1
03E6 008E           02136         movwf   DJNZ            ; save W in DJNZ for lo nibble writing
03E7 23F1           02137         call    Hinib_B         ; outputs w,4-7 to PORTB,4-7
03E8 23ED           02138         call    EnaLCD          ; generate enable signal for hi nibble
03E9                02139 Nibble
03E9 0E0E           02140         swapf   DJNZ,W          ; restore init value of W and swap it
03EA 23F1           02141         call    Hinib_B         ; outputs w,0-3 to PORTB,4-7
03EB 23ED           02142         call    EnaLCD          ; generate enable signal for low nibble
03EC 2BD0           02143         goto    loop130         ; wait 130 for LCD to crunch command
                    02144
                    02145 ;*****************************************************************
```

```
                    02146 ;* Generate Enable signal (1200us) for LCD controller,and refresh LEDs.
                    02147 ;* Entry point DisEna: Remove enable & dischg signal,and refresh LEDs.
                    02148 ;*****************************************************************
03ED                02149 EnaLCD
03ED 1406           02150         bsf     PORTB,0         ; enable LCD controller
03EE 2BEF           02151         goto    $+1             ; wait 2 cycles,make signal 1.2us long
03EF                02152 DisEna
03EF 1006           02153         bcf     PORTB,0         ; terminate enable signal LCD control
                    02154
                    02155 ;*****************************************************************
                    02156 ;* MoveLEDs
                    02157 ;*  Entry point MoveLEDs: transfer FLAG,LEDP FLAG,LEDH and FLAG,LEDL to
                    02158 ;*                        PORTB,5 PORTB,6 and PORTB,7 to service LEDs.
                    02159 ;*
                    02160 ;* Input variables:
                    02161 ;*    FLAG, bits LEDP, LEDH, LEDH will affect LED1, LED2, LED3
                    02162 ;* Output variables: none
                    02163 ;*****************************************************************
                    02164 ;* Hinib_B
                    02165 ;*  Entry point Hinib_B:  output W,4-7 to 4-bit LCD data bus
                    02166 ;*
                    02167 ;* Input variables:
                    02168 ;*    hi nibble of W is copied to LCD data bus
                    02169 ;* Output variables: none
                    02170 ;*****************************************************************
03F0                02171 MoveLESd                        ; writes states LEDs L,H,P to PORTB,5-7
03F0 080C           02172         movf    FLAG,W          ; FLAG bits 5,6,7 are LED bits
03F1                02173 Hinib_B                         ; outputs w,4-7 to PORTB,4-7
03F1 1206           02174         bcf     PORTB,4         ; clear high nibble of PORTB
03F2 1286           02175         bcf     PORTB,5         ; clear high nibble of PORTB
03F3 1306           02176         bcf     PORTB,6         ; clear high nibble of PORTB
03F4 1386           02177         bcf     PORTB,7         ; clear high nibble of PORTB
03F5 39F0           02178         andlw   0f0h            ; mask for hi nibble of W
03F6 0486           02179         iorwf   PORTB,F         ; write H nibble W to H nibble PORTB
03F7 0008           02180         return                  ; finished
                    02181
                    02182 ;*****************************************************************
                    02183 ;* HexDigit
                    02184 ;* This subroutine prints low nibble of W on LCD as hexadecimal digit.
                    02185 ;*  Zero (30h) is printed as capital O (7Fh)
                    02186 ;*
                    02187 ;* Input variables: W in range 0...0fh, hex number to be printed
                    02188 ;* Output variables: CHARCOU is decremented by two
                    02189 ;*****************************************************************
03F8                02190 HexDigit                        ; hex W (lo nibble) to LCD, change 0...
                    02191                                 ; ...to capital O
03F8 390F           02192         andlw   0fh             ; isolate low nibble of W...
03F9 009E           02193         movwf   BIN4            ; ...and put it in BIN4
03FA 3EF6           02194         addlw   -0ah            ; test if input number > 9
03FB 1C03           02195         btfss   STATUS,C        ; Is it > 9 ?
03FC 2B9B           02196         goto    NumBin4         ; ...if not, just print it as-is
03FD 3E41           02197         addlw   .7+3ah          ; 3ah...3fh to 'A'...'F' correction
03FE 2BD8           02198         goto    Char            ; print ASCII adjusted hex value a...f
                    02199
                    02200 ;*******  DATA EEPROM
                    02201 ;*****************************************************************
                    02202 ;*  This table is located in data eeprom
                    02203 ;*  It contains numerical data for 16 sample frequencies period display
                    02204 ;*  for analyzer. Last 13 bytes are timing constants used by subroutine
                    02205 ;*  GetSlowClk to generate internal timing (three fastest rates need no
                    02206 ;*  constants from the table, as they are treated as special cases)
                    02207 ;*****************************************************************
                    02208 ;*  ----- TABLE 1 (00h-2Fh): Rate display table for analyzer
                    02209 ;*
                    02210 ;*  ****First byte: Flags. Bits in this byte have the following functs:
                    02211 ;*  bit 7 = 0: Frequency in Hz
```

```
                    02212 ;*        = 1: Frequency in Mhz or Khz
                    02213 ;*  bit 6 = 0: Frequency does not contain decimal point
                    02214 ;*        = 1: Frequency contains decimal point before last digit
                    02215 ;*  bits  5,4: Bits 9 and 8 for frequency display, respectively
                    02216 ;*  bit 3 = 0: Period in us (microseconds)
                    02217 ;*        = 1: Period in ms (miliseconds)
                    02218 ;*  bit 2 = 0: Period does not contain decimal point
                    02219 ;*        = 1: Period contains decimal point before last digit
                    02220 ;*  bits  1,0: Bits 9 and 8 for period display, respectively
                    02221 ;*  **** Second byte: low significant byte for frequency
                    02222 ;*  **** Third byte: low significant byte for period
                    02223 ;*****************************************************************
                    02224 ;*  ----- TABLE 2 (30h-3Ch): Timing constant table for analyzer
                    02225 ;*
                    02226 ;*  Timing constant factors to all sample rates generated by subroutine
                    02227 ;*  GetSlowClk (all except 1MHz, 500KHz and 228KHz)
                    02228 ;*****************************************************************
                    02229 ;*  Note: This is read-only data, so the Data EEPROM must be programmed
                    02230 ;*  before the unit is ready to use. MCU will not affect data EEPROM
                    02231 ;*  contents. If your programmer does not support automatic loading of
                    02232 ;*  Data EEPROM contents from the HEX file, it must be loaded manualy.
                    02233 ;*  This will help in that case (all values are hexadecimal):
                    02234 ;*
                    02235 ;*  addr 00-07:  88 01 01 98 F4 02 8C E4
                    02236 ;*  addr 08-0f:  2C 88 64 0A 88 32 14 D8
                    02237 ;*  addr 10-17:  80 1A 88 19 28 C8 C0 34
                    02238 ;*  addr 18-1f:  88 0A 64 C8 60 68 C8 30
                    02239 ;*  addr 20-27:  D0 C9 18 A1 80 01 01 14
                    02240 ;*  addr 28-2f:  90 19 00 64 0A 00 28 19
                    02241 ;*  addr 30-37:  01 06 09 10 16 2E 30 64
                    02242 ;*  addr 38-3c:  CC 05 0E 3B 95
                    02243 ;*
                    02244 ;*  Total bytes used in Data EEPROM: 61 (the last 3 bytes don't care)
                    02245 ;*****************************************************************
2100                02246         org     2100h
                    02247 ;                                       constant T/sample Hz    s    RATE
                    02248
2100 0088 0001 0001 02249         de      b'10001000', .1,   .1   ; -     2.5      1M    1u    0
2103 0098 00F4 0002 02250         de      b'10011000', .244, .2   ; -     5        500K  2u    1
2106 008C 00E4 002C 02251         de      b'10001100', .228, .44  ; -     11       228K  4.4u  2
2109 0088 0064 000A 02252         de      b'10001000', .100, .10  ; 1     25       100K  10u   3
210C 0088 0032 0014 02253         de      b'10001000', .50,  .20  ; 6     50       50K   20u   4
210F 00D8 0080 001A 02254         de      b'11011000', .128, .26  ; 9     65       38.4K 26u   5
2112 0088 0019 0028 02255         de      b'10001000', .25,  .40  ; 16    100      25K   40u   6
2115 00C8 00C0 0034 02256         de      b'11001000', .192, .52  ; 22    130      19.2K 52u   7
2118 0088 000A 0064 02257         de      b'10001000', .10,  .100 ; 46    250      10K   100u  8
211B 00C8 0060 0068 02258         de      b'11001000', .96,  .104 ; 48    260      9.6K  104u  9
211E 00C8 0030 00D0 02259         de      b'11001000', .48,  .208 ; 100   520      4.8K  208u  10
2121 00C9 0018 00A1 02260         de      b'11001001', .24,  .161 ; 204   1040     2.4K  417u  11
2124 0080 0001 0001 02261         de      b'10000000', .1,   .1   ; 5     2500     1K    1m    12
2127 0014 0090 0019 02262         de      b'00010100', .144, .25  ; 14    6253     400   2.5m  13
212A 0000 0064 000A 02263         de      b'00000000', .100, .10  ; 59    25018    100   10m   14
212D 0000 0028 0019 02264         de      b'00000000', .40,  .25  ; 149   62548    40    25m   15
                    02265
                    02266 ;       timing constants table
2130 0001 0006 0009 02267         de      .001, .006, .009, .016, .022, .046, .048
     0010 0016 002E
     0030
2137 0064 00CC 0005 02268         de      .100, .204, .005, .014, .059, .149
     000E 003B 0095
                    02269
                    02270         end

All other memory blocks unused.

Program Memory Words Used:  1023
```



```
Program Memory Words Free:     1


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     4 reported,     0 suppressed
```

Make a Delta-Sigma Converter Using a Microcontroller's Analog Comparator Module

Authors: *Dieter Peter*
Bonnie C. Baker
Dan Butler
Hartono Darmawaskita
Microchip Technology Inc.

INTRODUCTION

This application note describes how to implement an Analog-to-Digital (A/D) Converter function using a member of the PIC16C6XX series of microcontrollers. Although these microcontrollers do not have a built-in A/D Converter like other controllers from Microchip, the comparator function, internal voltage reference and timers can be used to digitize an analog signal.

Some of the standard PICmicros have a comparator module, consisting of two comparators, both of which can be connected to PORTA in a variety of configurations. The internal voltage reference divider can be used with the comparators to establish thresholds. Additionally, one of the comparator inputs can be configured to the RA2 port allowing for the use of an external voltage reference. By combining these elements, a first order modulator and first order filter can be designed, emulating the function of an analog-to-digital delta-sigma conversion.

This method of conversion is quickly implemented in firmware with very few additional external components. Consequently, the cost of hardware implementation is minimal, particularly for such a high resolution converter solution. The input range is very flexible and adjusted with external resistors. Although this method is not particularly strong in terms of DC accuracy, it is well suited for ratiometic applications.

DELTA-SIGMA THEORY

The function of the classical Delta-Sigma Analog-to-Digital Converter is modeled with two circuit segments; a modulator and a digital filter. The modulator section acquires an input signal as shown in Figure 1. The input signal is added to a signal from a Digital-to Analog (D/A) Converter in the negative feedback loop. This differentiated signal then passes through an integrator and finally to one of the two inputs of a comparator. The comparator acts like a one-bit quantitizer. The output of the comparator is sent back to the differentiator via a one-bit Digital-to-Analog Converter. Additionally, the output of the comparator passes through a digital filter. With time, the output of the digital filter provides a multi-bit conversion result.

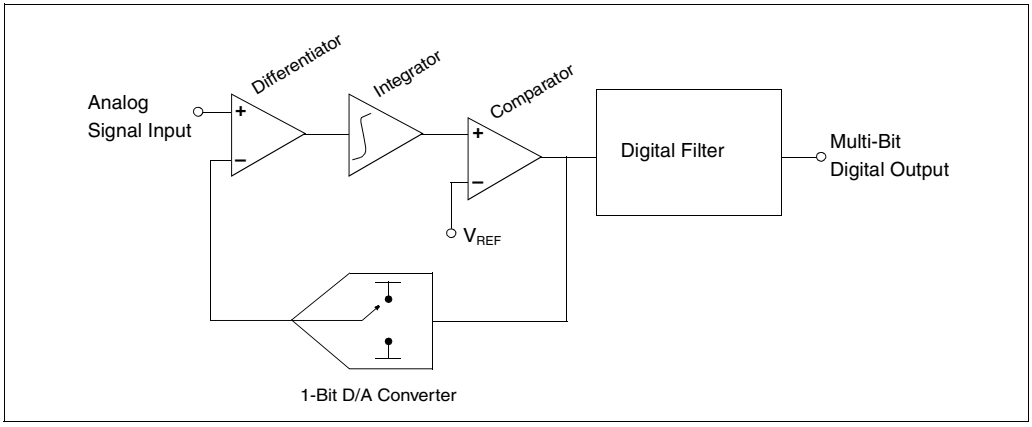


FIGURE 1: *First Order Delta-Sigma A/D Converter Block Diagram.*

This fundamental circuit concept has been used to generate a large variety of the converters that provide high resolution, relatively inexpensively. The next logical step for this type of A/D Converter is to move it into the controller. A basic controller is not able to execute this type of function, however, a few additional peripherals make it possible. The circuit diagram for this type of implementation is shown in Figure 2.

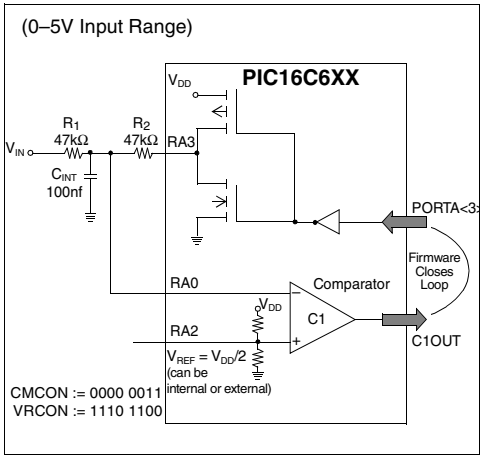


FIGURE 2: *A microcontroller can be configured as a Delta-Sigma Converter with two additional external resistors and one capacitor. In this configuration, a low pass filter is also implemented as part of the input network.*

In the circuit shown in Figure 2, the integrator function of the delta-sigma function is implemented with an external capacitor, C_{INT}. The absolute accuracy of this external capacitor is not critical, only its stability from integration to integration, which occurs in a relatively short period of time. When RA3 of the PIC16C6XX is set high, the voltage at RA0 increases in magnitude. This occurs until the output of the comparator (C1OUT) is triggered low. At this point the driver to the RA3 output is switched from high to low. Once this has occurred, the voltage at the input to the comparator (RA0) decreases. This occurs until the comparator is tripped high. At this point, RA3 is set high and the cycle repeats. While the modulator section of this circuit is cycling, two counters are used to keep track of the time and of the number of ones versus zeros that occur at the output of the comparator.

If this circuit were compared to the classical Delta-Sigma Converter, the integrator would be C_{INT}. The comparator is part of the controller, as well as its voltage reference. The one-bit D/A Converter is implemented in firmware by driving RA3 in accordance with the output of the comparator (CMCON<6>). The firmware drives the D/A Converter output at RA3. The digital filter is implemented with two counters.

IMPLEMENTATION WITH THE CONTROLLER

With the circuit in Figure 2, it is possible to conceptualize the delta-sigma function. The controller implementation of this circuit is summarized in the flow chart in Figure 3.

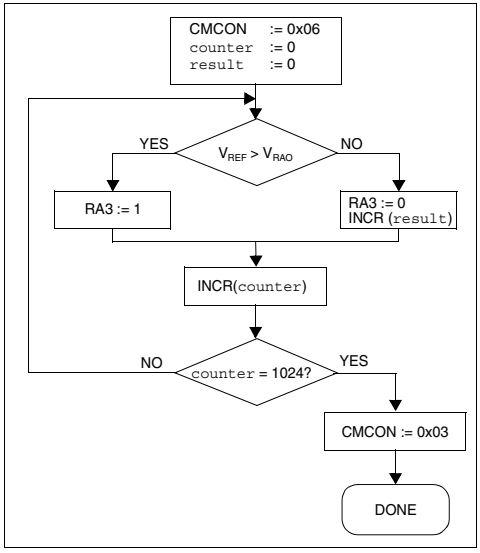


FIGURE 3: *A Delta-Sigma A/D Conversion Flow Chart implemented with circuit shown in Figure 2. Care should be taken to make the time required for every cycle taken through the flow chart to be a constant. This code is implemented until a conversion is complete.*

Normally the output of the comparator is directly connected to RA3 which keeps the voltage at RA0 equal to the reference voltage of the comparator in preparation for the next conversion.

When function “DeltaSigA2D” (Appendix A) is called to perform a conversion, the result and counter variables are cleared. Then the comparator is set to disconnect the output from RA3.This puts RA3 under active program control.

The comparator is checked at the beginning of each loop. If the voltage on the capacitor is less than the input voltage, RA3 is set high, which will put charge into the capacitor raising the voltage. If the voltage on the capacitor is greater than the input voltage, RA3 is set low, taking charge out of the capacitor lowering the capacitor voltage and the result register is incremented.

This continues as long as necessary to get the required resolution. For ten bits of resolution, 2^{10} (1024) laps through the loop are required. Each lap through the loop takes 17 instruction cycles. Padding is used to keep all paths through the code equal. A conversion cycle takes 17.5mS when using a 4 MHz clock.

When finished, the comparator output is fed directly to RA3, and the conversion is returned in result_l and result_h.

The sample code provided calls the DeltaSigA2D function and prints the result in an infinite loop. The output is transmitted at 9600 baud via RB7. The answers can be displayed on a dumb terminal program such as Hyperterm included with Windows '95.

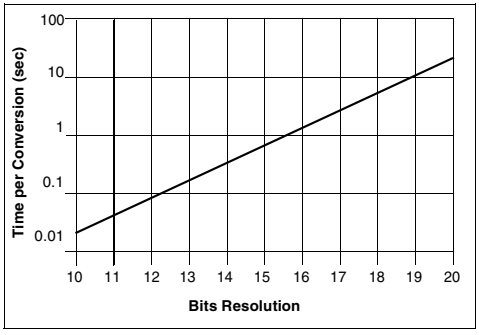


FIGURE 4: *Conversion time versus bits of resolution assuming a 20µs integration time.*

Each integration result is taken at a regular time interval. If it is assumed that the time interval of a conversion is 20µs, the conversion time versus bits can easily be calculated. This relationship is shown graphically in Figure 4. For instance, a 10-bit conversion would require 2^{10} or 1024 samples. If the microcontroller conversion loop is 20µs, one complete conversion would take 20.48ms.

Room temperature test data for the circuit shown in Figure 2 is graphed in Figure 5. In Figure 5, the voltage input is plotted versus the output code on the left axis and the output error on the right axis. This data was taken with the 1024 laps through the flow chart in Figure 3. The expected resolution of this configuration is 10-bits. The maximum code error for this test was ±2 counts or 2-bits of uncertainty. Consequently, the effective number of bits of this A/D Converter is 8-bits. The core portion of the code that was used to perform this conversion is listed at the end of the application note.

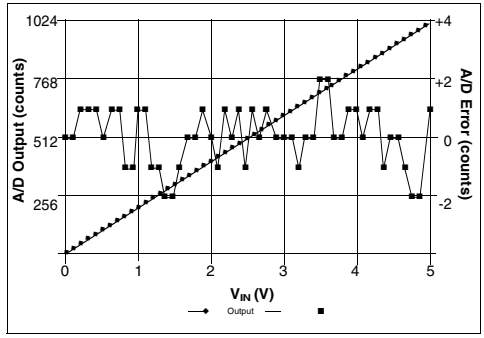


FIGURE 5: *Room temperature test data for the circuit shown in Figure 2. The input voltage range is 0.003 to 4.99V. The maximum error found in the test was ±2 counts. In this 10-bit system that is equivalent to ±9.8mV. This test was performed using one sample. Results may vary from part to part. VDD = 5V, calibration performed at 0.5V and 4.99V.*

The A/D error was calculated assuming the codes for Vin = 0.5V and Vin = 4.5V are ideal. This test was performed with one Microcontroller at room temperature. These result may vary from part to part.

ERROR ANALYSIS

This high resolution, low cost Delta-Sigma Converter provides a good solution for ratiometric applications where having the absolute results is not critical. Additionally, the function of analog gain is replaced by the inherent digital filtering that this technique utilizes.

In this example, V_{DD} is 5V and the reference voltage is ~V_{DD}/2. The resistors are 47kΩ, which are chosen to minimize the leakage errors across the resistors versus the RDS_{ON} error of the output pin, RA3. The capacitor has a value of 100nF.

RDS_{ON} Error

This error comes from the drain-source resistance of the output FETs on the output pin, RA3. At room temperature, this resistance error is typically less than 100Ω. Compared to R_2, RDS_{ON} introduces about 0.2% gain error. This is easily compensated for by increasing the resistor, R_1 by approximately 100Ω. Additionally, the value of the RDS_{ON} resistance will increase with rising temperature. Assuming a temperature change from 20°C to 70°C, RDS_{ON} will change from ~100Ω to ~200Ω which adds an additional 0.2% error.

RA0 Port Leakage Current

This leakage current is specified at 1nA at room temperature and 0.5µA (max) over temperature. The leakage current from the port at RA0 causes a voltage drop across the parallel combination of R_1 and R_2. With these two resistors equaling 47kΩ, the error caused by this leakage current is ~11mV. This is also close to a 0.2% error. At room temperature this error is negligible. Leakage current does increase with temperature.

Non-Symmetrical Output Port (RA3)

When the output port is high the FET resistance is dependent on the p-channel on resistance. When the output port is low the FET resistance is dependent on the n-channel on resistance. The p-channel on resistance is usually greater than the on resistance of the n-channel FET. As a consequence, there is an additional offset contribution of 5.5mV at room and over temperature.

Voltage Reference

The internal voltage reference to the comparator is implemented with a simple voltage divider. The absolute value of this voltage is dependent on internal resistor matching and power supply voltage. Assuming the power supply is an accurate 5V, the voltage error of this reference, part to part is significant. However, once the initial error of the internal voltage reference is removed with calibration, it is ratiometric to the power supply.

This is the biggest error in the circuit, but easily reduced with an external voltage reference.

Integration Capacitor

Any leakage errors of the capacitor will contribute to the overall error of the system. If the RC time constant of the circuit is greater than the sample frequency, the non-linearity of this time response will cause a linearity error in the system.

In this case the RC time constant is equal to:

$t_{RC} = R_1 || R_2 * C_{INT}$

$t_{RC} = 47k\Omega || 47k\Omega * 100nF$

$t_{RC} = 2.35ms$

The dielectric absorbtion is not critical. This is due to the fact that the capacitor voltage is held at a relative constant level.

In this example, the maximum voltage deviation due to the non-linearity of the RC network is ~8mV. This is also below a 0.2% error. If a lower sampling frequency is used, the integrating capacitor must be increased in value.

Comparator Offset

The offset of the comparator is specified at 10mV (max). With a V_{DD} of 5V, the error caused by the comparator is ~0.2%.

Error Source	Contribution at Room Temp		Error Due to Temperature	
	Offset	Gain	Offset	Gain
RDS_{ON} or RA3 (with R1 = 47kΩ+100Ω)	negligible	negligible		0.2%
Port Leakage	negligible	N/A	11mV	N/A
FET Symmetry of RA3	5.5mV	negligible	5.5mV	N/A
Internal Voltage Reference	49mV	N/A	49mV*	N/A
Comparator Offset	10mV	N/A	10mV	N/A
Most Probable Total Error			52mV*	0.2%

* the offset error of the internal voltage reference can be reduced significantly with an external reference.

TABLE 1: *Error contribution of all of the error sources at room and at temperature (-40 to 85°C) for R2 = 47kΩ. The "Most Probable Error Over Temperature" is calculated as the square root of the sum of the squares.*

Out of Range Inputs

In the event that the input signal goes to the maximum, minimum, or beyond the design limits, the converter will produce erroneous results. This problem can be corrected by decreasing R_2 by 10% to 20%.

Offset Adjustment

If the application requires that the effect of the system be nulled, this can be done by leaving V_{IN} open and running a conversion cycle. The results of this conversion will be equal to the offset voltage of the microprocessor system plus the external reference (if used).

OTHER INPUT RANGES

The configuration shown in Figure 2 is designed for a 0 to 5V input range. The input range for this circuit is determined by the resistor network (comprising of R_1 and R_2) and the reference voltage to the non-inverting input of the comparator. If the ratio of R_1 and R_2 is changed, the input range can be increased or decreased in accordance with the relationship between R_1 and R_2. Further adjustments can be implemented with an additional resistor added to this input structure that is biased to ground or the power supply.

Input Range of 2V to 3V

By adjusting the ratio of R_1 and R_2, the input range of this converter can be increased or decreased. The resistors that are selected for the circuit in Figure 6 reduces the input range from ±2.5V as in Figure 2 to +/-500mV. In both cases, the input range is centered around the reference voltage to the comparator, 2.5V. This type of input range is best suited for sensors with smaller output voltage ranges, such as the buffered output of a pressure sensor or load cell.

The resistors are determined by comparing the desired input range to the voltage range of RA3. Assuming that the reference voltage in this problem is 2.5V, the input range changes +/-500mV and the voltage at RA3 changes by +/-2.5V. The ratio of these two voltage ranges is 5:1. Consequently, during one integration period the difference between the current through R_2 and R_1 must always be less than zero. In this manner, the RA3 gate will be capable of driving the capacitor, C_{INT}, past the reference voltage applied to the non-inverting input of the comparator.

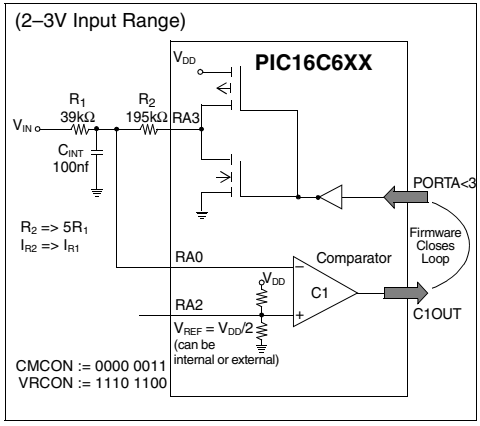


FIGURE 6: *Configuration of the microcontroller for a delta-sigma conversion with a ±500mV range centered around 2.5V.*

The design equations for this circuit are:

$$V_{IN(CM)} = V_{RA0}$$

$$V_{IN(P\ TO\ P)} = V_{RA3(P\ TO\ P)}\ (R_1/R_2)$$

where

$V_{IN(CM)}$ is equal to

$(V_{IN\ (MAX)} - V_{IN\ (MIN)})\ /2 + V_{IN\ (MIN)}$

V_{RA0} is the voltage applied to the comparator's inverting input

$V_{IN\ (P\ TO\ P)}$ is equal to $(V_{IN(MAX)} - V_{IN(MIN)})$

$V_{RA3\ (P\ TO\ P)}$ is equal to $V_{RA3(MAX)} - V_{RA3(MIN)}$

Input Range of 10V to 15V

By adding an additional resistor to the input structure of the A/D Converter, an offset adjustment can be applied to the input range. In Figure 7, R_1 and R_2 are equal and configured to allow for an input range of +/-2.5V as shown in Figure 2. The addition of R_3, which is referenced to ground, provides a level shift to the input range of 10V.

With this circuit configuration, a 5V (full-scale) current through R_1 is equal to V_{REF} / R_1. If R_3 is used to draw the same current to ground, the integrating capacitor will not be charged. In this manner, a 2.5V offset is implemented with $R_3 = R_1$. To achieve a 10V offset, R_3 must be equal to $4*R_1$ as shown in Figure 7.

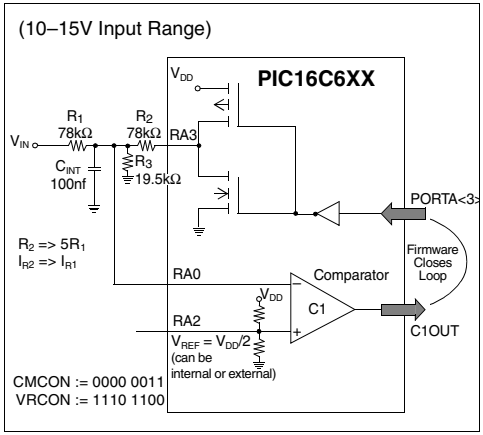


FIGURE 7: *Configuration of the microcontroller for a delta-sigma conversion with a ±2.5V range centered around 12.5V.*

The design equations for this circuit are:

$$V_{IN(CM)} = V_{RA0}\ (1 + R_1/R_3)$$

$$V_{IN(P\ TO\ P)} = V_{RA3(P\ TO\ P)}\ (R_1/R_2)$$

where

$V_{IN(CM)}$ is equal to

$(V_{IN\ (MAX)} - V_{IN\ (MIN)})\ /2 + V_{IN\ (MIN)}$

V_{RA0} is the voltage applied to the comparator's inverting input

$V_{IN\ (P\ TO\ P)}$ is equal to $(V_{IN(MAX)} - V_{IN(MIN)})$

$V_{RA3\ (P\ TO\ P)}$ is equal to $V_{RA3(MAX)} - V_{RA3(MIN)}$

Input Range of ±500mV

The circuit in Figure 8 using the scaling technique discussed in the circuit shown in Figure 5 and the offset shift technique discussed in the circuit shown in Figure 6. With this circuit, the input range is +/-500mV. This is achieved by making $R_2 = 5R_1$. Then the signal input range is level shifted by -2.5V. In the circuit in Figure 8 this is implemented with a resistor, R_3, to the positive supply. This level shift is achieved by making $R_3 = R_1$.

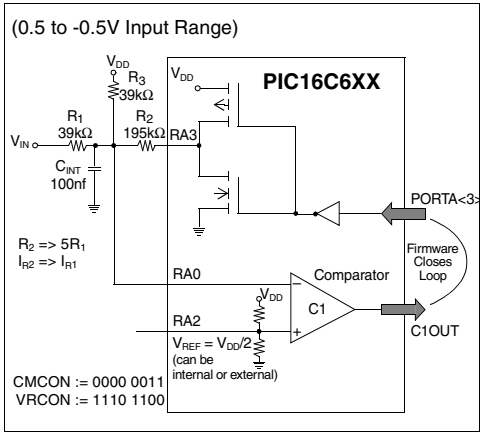


FIGURE 8: *Configuration of the microcontroller for a delta-sigma conversion with a ±500mV range centered around ground.*

The design equations for this circuit are:

$$V_{IN(CM)} = V_{RA0}\ (1 + R_1/R_3)$$

$$V_{IN(P\ TO\ P)} = V_{RA3(P\ TO\ P)}\ (R_1/R_2)$$

where

$V_{IN(CM)}$ is equal to

$(V_{IN\ (MAX)} - V_{IN\ (MIN)})\ /2 + V_{IN\ (MIN)}$

V_{RA0} is the voltage applied to the comparator's inverting input

$V_{IN\ (P\ TO\ P)}$ is equal to $(V_{IN(MAX)} - V_{IN(MIN)})$

$V_{RA3\ (P\ TO\ P)}$ is equal to $V_{RA3(MAX)} - V_{RA3(MIN)}$

This circuit can be used to measure the current through a shunt resistor. The main error term at room temperature is comparator offset. In systems with a known "zero-current" state, the offset can be measured and removed through calculation or removed by adding or subtracting the offset to the `result` counter.

REFERENCES

Cox, Doug, "Implementing Ohmmeter/Temperature Sensor", AN512, Microchip Technology, Inc.

Richey, Rodger, "Resistance and Capacitance Meter Using a PIC16C622", AN611, Microchip Technology, Inc.

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX A: SOURCE CODE = DELTASIG.ASM

```
;*****************************************************************
;* Filename: DeltaSig.asm
;*****************************************************************
;* Author:          Dan Butler
;* Company:         Microchip Technology Inc.
;* Revision:        1.00
;* Date:            02 December 1998
;* Assembled using MPASM V2.20
;*****************************************************************
;* Include Files:
;*          p16C622.inc     V1.01
;*****************************************************************
;* Provides two functions implementing the Delta Sigma A2D.
;* InitDeltaSigA2D sets up the voltage reference and comparator
;* in the "idle" state.
;* DeltaSigA2D runs the actual conversion.  Results provided in
;* result_l and result_h.
;*          See An700 figure 2 for external circuitry required.
;*****************************************************************
;* What's changed
;*
;* Date    Description of change
;*
;*****************************************************************
#include   <p16C622.inc>
cblock
           result_l
           result_h
           counter:2
endc
;
;
;
InitDeltaSigA2D
           bsf         STATUS,RP0
           movlw       0xEC
           movwf       VRCON
           bcf         PORTA,3         ;set comparator pin to output
           bcf         STATUS,RP0
           movlw       0x06            ;set up for 2 analog comparators with common reference
           movwf       CMCON
           return
;
;   Delta Sigma A2D
;   The code below contains a lot of nops and goto next instruction.  These
;   are necessary to ensure that each pass through the loop takes the same
;   amount of time, no matter the path through the code.
;
DeltaSigA2D
           clrf        counter
           clrf        counter+1
           clrf        result_l
           clrf        result_h
           movlw       0x03            ; set up for 2 analog comparators with common reference
           movwf       CMCON
loop
           btfsc       CMCON,C1OUT     ; Is comparator high or low?
           goto        complow         ; Go the low route
comphigh
           nop                         ; necessary to keep timing even
           bcf         PORTA,3         ; PORTA.3 = 0
           incfsz      result_l,f      ; bump counter
```

```
            goto        eat2cycles      ;
            incf        result_h,f      ;
            goto        endloop         ;
complow
            bsf         PORTA,3         ; Comparator is low
            nop                         ; necessary to keep timing even
            goto        eat2cycles      ; same here
eat2cycles
            goto        endloop         ; eat 2 more cycles
endloop
            incfsz      counter,f       ; Count this lap through the loop.
            goto        eat5cycles      ;
            incf        counter+1,f     ;
            movf        counter+1,w     ;
            andlw       0x04            ; Are we done? (We're done when bit2 of
            btfsc       STATUS,Z        ; the high order byte overflows to 1).
            goto        loop            ;
            goto        exit
eat5cycles
            goto        $+1             ; more wasted time to keep the loops even
            nop                         ;
            goto        loop            ;
exit
            movlw       0x06            ; set up for 2 analog comparators with common reference
            movwf       CMCON
            return
            end
```

Switch Mode Battery Eliminator Based on a PIC16C72A

Author: *Brett Duane*
Microchip Technology

OVERVIEW

The PIC16C72A is a member of the PICmicro® Mid-Range Family of 8-bit, high-speed microcontrollers. The PIC16C72 provides the following features:

- 5 Channel, 8-bit Analog-to-Digital Converter (A/D)
- CCP Module to generate a PWM output
- I^2C™/SPI™ Module
- 3 Timers
- 8 Interrupt sources

This application note shows how to combine the A/D and CCP modules with suitable software to produce a Switch Mode Battery Eliminator (SMBE) providing 3.0, 4.5, 5.0, 6.0, 7.5 and 9.0 volt output voltages at up to 1 Amp with an AC or DC input between 12.6V and 30V peak.

HARDWARE

The system makes use of the A/D to read the input and output voltages, the Capture/Compare/Pulse module to generate a PWM output, and Timer2 to regulate how fast the program runs. External hardware includes a switching power converter and a suitable output filter. Six LEDs on PORTB indicate the output voltage as set by two push buttons on PORTA.

Optional components not installed in this project include a serial EEPROM to store the last voltage setting and a level translator to convert TTL to RS-232 for communications with a PC.

In-Circuit Serial Programming™ (ICSP) support has also been provided. LEDs D7 and D8 share clock and data lines required for ICSP. These LEDs indicate error conditions and are optional.

Analog-to-Digital Converter Module

The A/D converts an input voltage between ground and V_{DD} to an 8-bit value presented in ADRES. In this application, the switching converter input and output voltages are sampled. Provisions have been included to read the setting of a potentiometer.

Capture/Compare/Pulse Width Modulation Module

The CCP module produces the PWM signal that controls the series pass switching transistor. Depending on the PWM period and F_{OSC}, any number of bits between 2 and 10 bits may be used to specify the PWM on-time. The CCP module requires the use of Timer2, the Timer2 prescaler, and the PR2 register to produce a PWM output.

Timer2 Postscaler

Timer2 drives the CCP module to control the PWM period and also drives the Timer2 postscaler. The postscaler is incremented when Timer2 is reset at the end of each PWM cycle and will generate an interrupt when the postscaler overflows.

External Hardware

The switching buck converter relies on three components to function:

- Series pass switch (Q1)
- Inductor (L1)
- Commutating diode (D10).

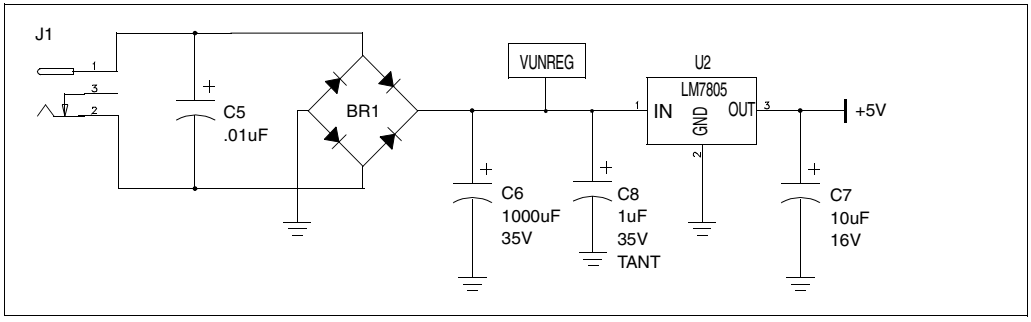
These devices form the core of all switch mode buck converters.

Power Input Circuit

This circuit is a conventional linear power supply that accepts AC or DC power with a peak voltage of 30V (limited by the 78L05). The converter operates off the unregulated bulk power, while the regulator supplies power and a voltage reference to the controller.

Figure 1 shows the Power Input Circuit. Bridge rectifier BR1 rectifies the raw power input that may be AC or DC. Capacitor C6 provides rough filtering to reduce ripple in the input voltage. Capacitor C8 provides the short current pulses drawn by transistor Q1 when it is turned on. Capacitor C7 provides filtering of regulator U2 output.

FIGURE 1: POWER INPUT CIRCUIT



Power Converter

Figure 2 shows the switching buck converter with drive circuits. Unregulated DC is provided at the emitter of transistor Q1. Q1, inductor L1 and diode D10 form the basic buck switching converter. The output appears at connector J4.

When RC2 (PWM output) is high, transistor Q2 is turned on, pulling Q2's collector to ground. This draws current from Q1's base, turning Q1 on. When Q1 is on, current from capacitors C6 and C8 charge L1 through the load. Resistor R19 limits the current drawn from the base of Q1. Resistor R17 ensures that Q1 switches off quickly. Resistor R20 ensures that transistor Q2 switches off quickly. Resistor R18 limits Q2's base current.

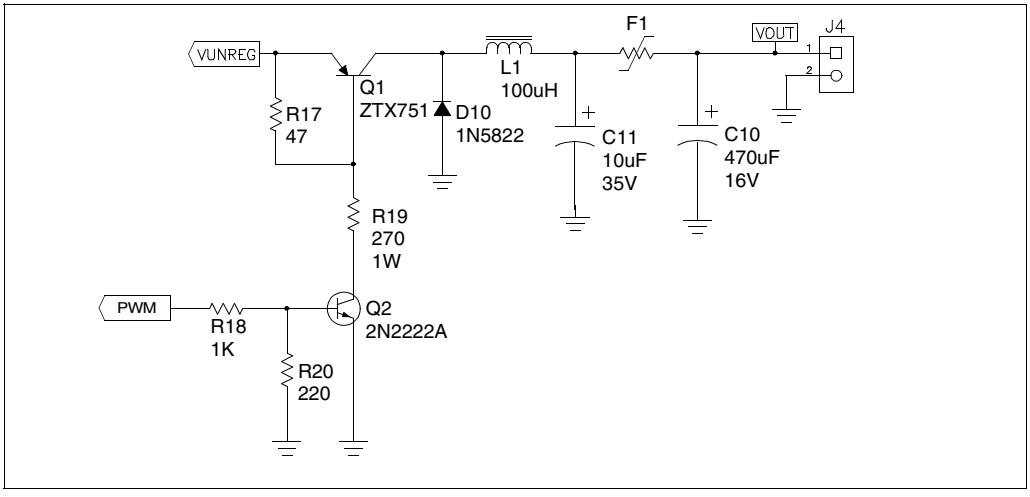
When RC2 is low, R20 turns off Q2 and R17 turns off Q1. When Q1 is off, the current through L1 continues to flow through L1, D10 and the load, discharging L1. When the current through L1 becomes less than the current drawn by the load, C10 provides the additional current and reduces output voltage ripple.

To ensure that Q1 remains cool during operation, it must be driven well into saturation. When driving transistors as digital switches, divide their h_{FE} (small signal gain) by 5 and use the resulting gain for your calculations. This ensures that the transistor switches through its linear region quickly to prevent significant heat generation in the transistor.

As the unregulated input voltage decreases, the drive applied to Q1 decreases to the point where Q1 starts operating in its linear region, producing heat. Continued operation in the linear region will cause Q1 to overheat and fail. Q1 usually shorts, causing the input voltage to appear at the converter output. R19 was selected to allow Q1 to operate safely as long as VUNREG is above 10V. The controller software will shut down the converter if VUNREG falls below 10V.

The converter output contains a considerable amount of noise. Capacitors C10 and C11 provide filtering to reduce that noise. F1 is a PTC resistor acting as a 1 Amp, self-resetting fuse.

FIGURE 2: SWITCHING BUCK CONVERTER WITH DRIVE AND OUTPUT CIRCUITS



Microcontroller Circuits

The microcontroller, analog inputs and digital outputs are shown in Figure 3.

The LEDs indicate the output voltage (D1-6) and converter faults (D7-8). Switches S1 and S2 allow the user to select the desired output voltage. Resistors R3, R4 and capacitor C4 form the voltage feedback circuit. Resistors R15, R16 and capacitor C13 form the voltage source sense circuit. PWM is output at pin RC2.

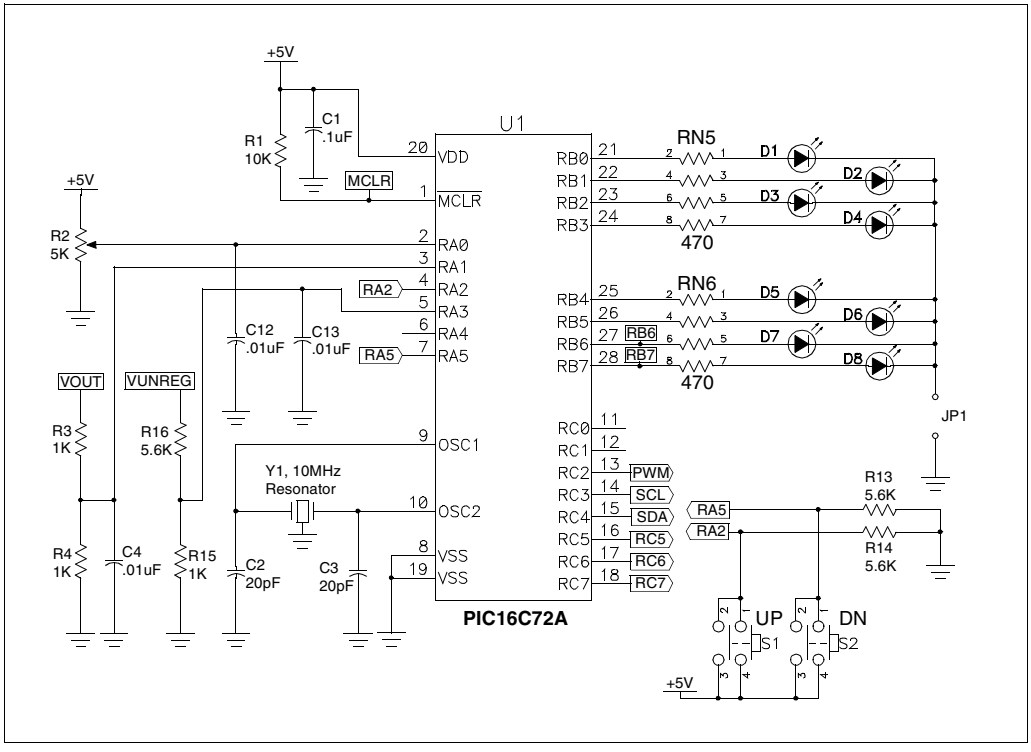
Register packs RN5 and RN6 limit current through LEDs D1-D8. LEDs D7 and D8 share clock and data lines required for ICSP. Jumper J1 disables all LEDs.

When programming the controller using ICSP, J1 should be removed. If ICSP does not function properly, D7 and D8 or RN6 should be installed after programming.

S1 is the decrease voltage button and S2 is the increase voltage button. R13 and R14 are pull down resistors.

Resonator Y1, and capacitors C2 and C3 set Fosc for the controller. If Y1 is a ceramic resonator with internal capacitors, C2 and C3 are not required. Resistor R1 pulls $\overline{\text{MCLR}}$ to +5V while allowing the controller to be programmed using ICSP. In addition, connecting pins 1 and 3 of connector J2 causes a remote reset of the controller. (See figure 5.)

FIGURE 3: MICROCONTROLLER, ANALOG INPUTS AND DIGITAL INPUTS AND OUTPUTS



FIRMWARE

Initialization

The controller is first initialized by configuring the A/D, CCP and Timer2 peripherals, followed by clearing the RAM required for variables and initializing some variables. Controller pins are configured as required for each of the modules, or as digital outputs if they are not being used.

A/D

Pins RA0, RA1 and RA3 are configured as analog inputs. Pins RA2 and RA5 are used as digital inputs. The controller VDD is used as VREF for the A/D.

The conversion clock source TAD is selected to be between 1.6usec and 6.4usec. Since Tosc = 0.1usec (Fosc = 10MHz), 32Tosc = 3.2usec. The A/D module is turned on and pin RA1 (VOUT) is sampled for conversion later in the loop.

TRISA configures pin RA4 as an output and all others as inputs.

CCP (PWM)

The CCP module is set to PWM mode. Timer2 is enabled with a 1:1 prescaler. PR2 is set to 63 (0x3F). The resulting PWM frequency is 39.063KHz. (TPWM=25.6msec). The CCP module uses 8-bit data to control the PWM duty cycle. The PWM duty cycle is set to 0, ensuring that the PWM output is turned off.

All pins on PORTC are configured as outputs, including RC2 which is the controller PWM output.

Timer2 postscaler

Since Timer2 will also control the frequency that the main loop will execute, the Timer2 postscaler is set to a 1:1 ratio and the Timer2 postscaler interrupt is enabled. This will cause one interrupt for each PWM cycle.

RAM

RAM required for variables is cleared. The 3.0V LED on PORTB is lit. The variable `set_pt` is initialized to produce the 3.0V output. Button debounce counters are initialized.

Main loop

For a digital control loop to function as well as an analog controller, the digital control loop should repeat at least 30 times faster than the fastest expected transient.

The ripple frequency at capacitor C7 is 120Hz, or twice the AC power line frequency. In this application, the loop must be executed at least 120 x 30, or 3600 times a second to adequately respond to the bulk power ripple. Transients at the converter output are also handled, but less effectively with increasing frequency.

The Timer2 postscaler generates interrupts that are counted by the interrupt service routine. When 8 interrupts have occurred, the main loop is allowed to execute once. This causes the main loop to execute 4883 times a second.

The A/D starts a conversion on RA1/AN1 (VOUT). The program loops wait for the conversion to complete. The 8-bit result is placed in VOUT. The A/D is then set to sample the input voltage (VUNREG).

PID Controller

The control algorithm is a software implementation of a Proportional-Integral-Differential (PID) controller. The only input to this controller is the difference between the desired output voltage and the actual output voltage, and is known as the error signal.

The first module produces the error signal by finding the difference between `set_pt` and VOUT. The result is saved in the low byte of a 2 byte signed variable `e0h:e0`. The high byte is set to reflect a negative value if needed. The difference is selected to produce a positive result if `set_pt` is greater than VOUT.

```
e0 = set_pt - vout
```

if `e0<7>=0` , `e0h = 0x00`, else `e0h = 0xFF`

A proportional term is generated from the present error signal. This is simply the signed error multiplied by some factor `Kp`. The 2 byte signed result is saved as `proh:pro`.

```
proh:pro = Kp * e0h:e0
Kp = proportional factor
```

The difference between the present and previous error is found and saved as the 2 byte signed result `difh:dif`. The previous error `e1h:e1` is simply the error result found in the execution of the previous loop. The difference between errors is multiplied by some factor (`Kd`) and saved as `difh:dif`.

```
difh:dif = Kd * (e1h:e1 - e0h:e0)
Kd=difference factor
```

The integral component is nothing more than a total of all the errors produced since the last reset. The present error is multiplied by some factor (Ki) before being added to the running 2 byte unsigned total `inth:int`.

```
inth:int = inth:int + Ki * e0h:e0
Ki = integral factor
```

The proportional, integral and difference terms are summed together. The result of the sum is saved as the 2 byte signed result `pwmh:pwm`.

```
pwmh:pwm = proh:pro + difh:dif
             + inth:int
```

The result is never greatly positive, but can sometimes cause the result to underflow. When pwmh:pwm underflows (as it does when the load is disconnected), the PWM drive signal must be forced to zero or the converter output becomes unpredictable. The overload LED is also lit.

If pwmh<7> = 1, then pwmh:pwm = 0x0000 and turn on the OVLD LED, else turn off OVLD LED.

PWM Generator

The PWM generator module (software, not the peripheral) discards the 3 least and 5 most significant bits, and uses the remaining 8 bits to generate PWM with a desired on time. Of the remaining 8 bits, the 2 least significant bits are loaded into CCP1CON<5:4>. The last 6 bits (with 2 leading zeros to form a byte) are loaded into CCP1RL.

Conversion of the input voltage V_{UNREG} is started.

Reading Buttons

The LED data at PORTB is copied to a temporary variable.

The down button is read. If it is closed, its debounce counter is incremented. If no overflow occurs, execution proceeds to reading the up button. If an overflow does occur, the LED data is shifted right one bit, unless bit 0 is already set. Overflows occur approximately every half second.

The up button is read and processed similar to the down button. If it is closed, its debounce counter is incremented. If no overflow occurs, execution proceeds to convert VIN. If an overflow does occur, the LED data is shifted left one bit, unless bit 5 is already set. Overflows occur approximately every half second.

The LED data is copied back to PORTB to light the corresponding LED. The LED data is also used to find the proper index to use prior to calling a look-up table.

A call to the lookup table is performed. The lookup table routine adds the index in the W register to the program counter. The next instruction performed is a "retlw" instruction that places the new set_pt in the W register and returns to the next instruction after the call to the look-up table. The value returned is saved as the new set_pt.

Both button debounce counters are reset.

The conversion result of V_{IN} is retrieved from the A/D and vin is subtracted from 0xC1 (10V set point). If the result is positive, V_{IN} is less than 10V and program execution is directed to a safety shutdown module. Otherwise, program execution continues normally.

```
If (0xC1 - vin) is positive, go to shutdown.
```

The present error, e0h:e0, replaces the previous error, e1h:e1.

Program execution then returns to the top of the loop to wait for Timer2 postscaler interrupts.

Interrupts

The Interrupt Service Routine saves the state of the W and STATUS registers, increments the interrupt counter and restores the STATUS and W registers.

Safety Shutdown

The safety shutdown module has been included to turn off the converter and to light the trip LED. Once entered, there is no exit from this module except by resetting the controller.

Gain constants Kp, Kd, and Ki

Constants Kp, Ki, and Kd were determined experimentally. The goal was to maintain VOUT between 4.75V and 5.25V when the 5V output was selected. VOUT should remain within this band, regardless of changes in the load current. This specifically includes 10% changes in current, unloaded to full-load, and full-load to unloaded step changes. Other step changes in loading were also examined.

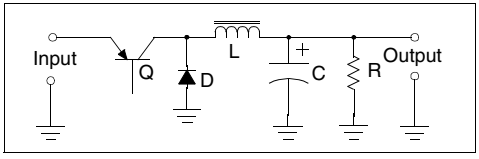
Constants determined for a 5V output were then used for other voltage outputs.

Only resistive loads were considered. Some degradation of output performance may be expected with inductive or capacitive loads.

APPENDIX A: SWITCH MODE BUCK CONVERTER

A switch mode buck converter performs a voltage reduction by periodically charging an inductor from a high voltage source, then allowing the charged inductor to transfer its stored energy to the load at a lower voltage. This energy transfer occurs with little loss.

FIGURE 4: SWITCH MODE BUCK CONVERTER TOPOLOGY



In the ideal buck converter, the average output power equals the average input power. The switch is operated at a constant frequency, but the duty cycle (on time / cycle time) controls the output voltage.

When the switch Q (a transistor or MOSFET) is closed, current from the source flows through the inductor L, through the load R and back to the source. Energy is being stored in the inductor by increasing inductor current and building up a magnetic field.

When the switch is open, the source no longer supplies energy. The energy stored in the magnetic field is transferred to the load as the magnetic field collapses. Inductor current is decreasing as current flows from the inductor through the load R and diode D and back to the inductor L.

No energy is lost in the inductor during this conversion. The majority of losses occur in the switching device as it switches though its linear region, and the commutation diode when it conducts due to its forward voltage drop. Most electrolytic filter capacitors also have high resistance to the high frequency ripple current.

If current flows through the inductor at all times, the converter is operating in the continuous mode. The average inductor current is the same as the load current. If the load current is constant, variations in the inductor current average to zero. The peak inductor current will be no greater than twice the average current, and can be reduced by raising the switching frequency. This is normally the desired operating mode.

If the current through the inductor stops, the converter is operating in the discontinuous mode. All the current that flows through the load continuously must flow through the switch and charge the inductor during the time the switch is turned on. This can result in very high currents in both the switch and inductor and risks saturating the inductor. When the inductor is saturated, its inductance decreases drastically. Considerable noise is also produced as large currents are switched, requiring a greater amount of filtering. This mode is normally used only when there is very light loading of the converter, but it is easier to stabilize than the continuous mode converter.

In both continuous and discontinuous modes, charging or discharging a filter capacitor at the converter output makes up the difference between the inductor and load currents. This filter capacitor also reduces output ripple voltage and noise that switching converters produce.

The equations presented here assume an ideal circuit, but actual circuits have similar results.

The approximate duty cycle D.C. can be approximated by:

$$D.C. = \frac{V_{OUT}}{V_{IN}} = \frac{I_{IN}}{I_{OUT}} = \frac{T_{ON}}{T_{PWM}}$$

Where V_{OUT} = output voltage,
I_{IN} = average input current,
T_{ON} = switch on time
V_{IN} = input voltage,
I_{OUT} = output current,
T_{PWM} = PWM period = 1/F_{PWM}

The ripple current magnitude due to switching is approximated by:

$$I_{RIPPLE} = \frac{(V_{IN}-V_{OUT}) * D.C. * T_{PWM}}{L}$$

Where L = inductor inductance,
I_{RIPPLE} = ripple current peak-to-peak
F_{PWM} = PWM frequency

The ripple current is absorbed by the output filter capacitor C, but produces a small output ripple voltage that is approximated by:

$$V_{RIPPLE} = \frac{I_{RIPPLE} * D.C. * T_{PWM}}{C}$$

Where V_{RIPPLE} = output voltage ripple peak-to-peak

EXAMPLE 1: CAPACITOR AND INDUCTOR SELECTION

Given: V_{UNREG} = 20V, V_{OUT} = 6V,
V_{RIPPLE} = 0.1V, I_{RIPPLE} = 1A,
F_{PWM} = 39.063KHz

Solution:

$$D.C. = \frac{V_{OUT}}{V_{IN}} = \frac{6}{20} = 0.30$$

$$T_{PWM} = \frac{1}{F_{PWM}} = \frac{1}{39KHz} = 25.6uS$$

$$C = \frac{I_{RIP}*D.C.*T_{PWM}}{V_{RIPPLE}} = \frac{(1A)(.3)(25.6uS)}{0.1V}$$

$$= 76.8uF$$

$$I_{RIP} = \frac{(V_{IN}-V_{OUT})*D.C.*T_{PWM}}{L}$$

$$L = \frac{(V_{IN}-V_{OUT})*D.C.*T_{PWM}}{I_{RIP}}$$

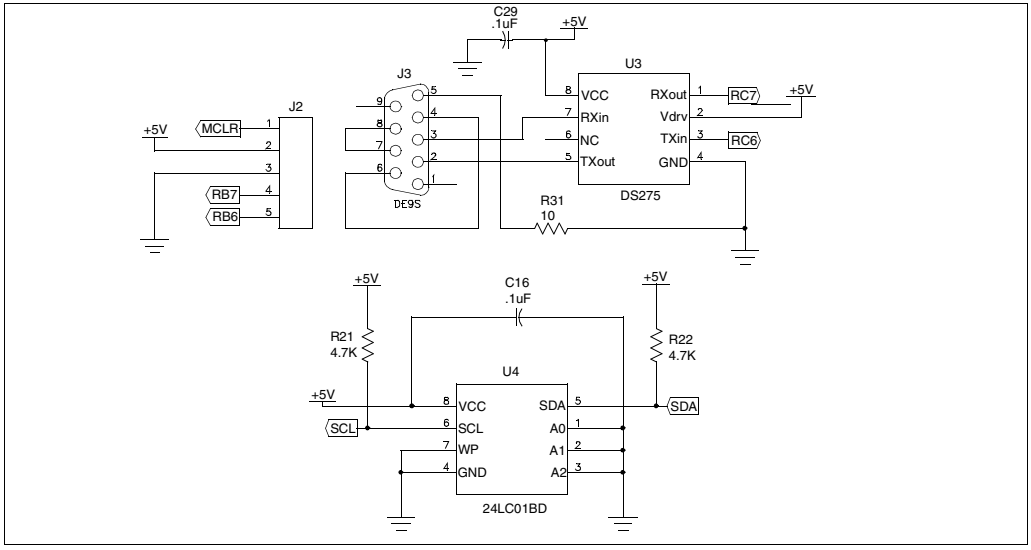
$$= \frac{(20V-6V)(0.30)(25.6uS)}{1A}$$

$$= 107.5uH$$

APPENDIX B: ACCESSORY COMPONENTS

These accessory component are not installed on the board, but can provide additional capabilities. Connector J2 provides for In-Circuit Serial Programming (ICSP) and offers a remote MCLR reset by connecting pins 1 and 3. Integrated circuit U3 and connector J3 provide RS-232 communications with the controller. Integrated circuit U4 is a serial EEPROM that communicates with the controller using the I^2C™ protocol.

FIGURE 5: ACCESSORY COMPONENTS



Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX C: CODE

```
MPASM 02.20 Released         SW_REG1A.ASM   3-9-1999  19:28:40


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001
;*******************************************************************************
                     00002 ;*   Filename: APPNOTE.ASM
                     00003
;*******************************************************************************
                     00004 ;*       Author:      Brett Duane
                     00005 ;*       Company:     Microchip Technology
                     00006 ;*       Revision:    Rev 1.0
                     00007 ;*       Date:        3-9-99
                     00008 ;*       Assembled using MPASM rev 4.00.00
                     00009
;*******************************************************************************
                     00010 ;*       Include files: p16c72a.inc Rev 1.01
                     00011
;*******************************************************************************
                     00012 ;*
                     00013 ;*       Switching buck regulator using 16C72A using A/D,
                     00014 ;*       PWM (CCP) and timer2 modules.
                     00015 ;*
                     00016 ;*       PID control loop implementation.
                     00017 ;*       Executes 4883 loops per second.
                     00018 ;*       Spends most of its time looping waiting for timer2 overflows.
                     00019 ;*
                     00020 ;*       A/D inputs and PWM output are 8 bits,
                     00021 ;*       with internal calculations done in 16 bits
                     00022 ;*
                     00023 ;*       Timer2 postscaler (1:16) generates an interrupt.
                     00024 ;*
                     00025 ;*       RA0 converts a 0-5V input from trimmer, but is not used.
                     00026 ;*       RA1 monitors VOUT.
                     00027 ;*       RA2 is the up voltage push button input.
                     00028 ;*       RA3 monitors VUNREG.
                     00029 ;*       RA5 is the down voltage push button input.
                     00030 ;*
                     00031 ;*       RB<5:0> drives LEDs to indicate the output voltage.
                     00032 ;*       RB<7:6> drives LEDs to indicate overload or shutdown.
                     00033 ;*
                     00034 ;*       RC2 is the PWM source for the switching converter.
                     00035 ;*
                     00036 ;*
                     00037 ;*       Configuration Bit Settings
                     00038 ;*           Brown-out detect is off
                     00039 ;*           Code protect is off
                     00040 ;*           Power-up timer is enabled
                     00041 ;*           Watchdog timer is disabled
                     00042 ;*           10MHz resonator is driven in XT mode
                     00043 ;*
                     00044 ;*       Program Memory Words Used:   230
                     00045 ;*       Program Memory Words Free:  1818
                     00046 ;*
                     00047 ;*       Data Memory Bytes Used:       24
                     00048 ;*       Data Memory Bytes Free:      104
                     00049 ;*
                     00050
;*******************************************************************************
```

```
                  00051 ;*       What's Changed
                  00052 ;*
                  00053 ;*       Date        Description of Change
                  00054 ;*
                  00055 ;*   3-3-99      This is the initial release.
                  00056 ;*
                  00057
;*****************************************************************************
                  00058
                  00059      list p=16c72a
                  00060      #include    <p16c72a.inc>
                  00001      LIST
                  00002 ; P16C72A.INC Standard Header File,Version 1.01 Microchip Technology, Inc.
                  00249      LIST
                  00061
2007 3FB1         00062      __config _BODEN_OFF & _CP_OFF & _PWRTE_ON & _WDT_OFF & _XT_OSC
                  00063
                  00064
                  00065          cblock  0x020
                  00066
  00000020        00067              UPCL     ;up button debounce
  00000021        00068              UPCH     ;up button debounce
  00000022        00069              DNCL     ;down button debounce
  00000023        00070              DNCH     ;down button debounce
  00000024        00071              SETPOINT ;voltage setpoint - RA0 result (unsigned)
  00000025        00072              VOUT     ;output voltage feedback - RA1 result
  00000026        00073              VUNREG   ;source voltage feedback - RA3 result
  00000027        00074              TEMPA    ;temp variable
  00000028        00075              TEMPB    ;temp variable
  00000029        00076              INT      ;integral component
  0000002A        00077              INTH     ;integral component
  0000002B        00078              PRO      ;proportional component
  0000002C        00079              PROH     ;proportional component
  0000002D        00080              DIF      ;difference component
  0000002E        00081              DIFH     ;difference component
  0000002F        00082              PWM      ;PWM drive
  00000030        00083              PWMH     ;PWM drive
  00000031        00084              E0       ;present error
  00000032        00085              E0H      ;present error
  00000033        00086              E1       ;past error
  00000034        00087              E1H      ;past error
  00000035        00088              T2POST   ;postscaler interrupt counter
  00000036        00089              ISRS     ;isr variable
  00000037        00090              ISRW     ;isr variable
                  00091
                  00092          endc
                  00093
  000000F9        00094 DEL1     equ 0xf9     ;text equate - debounce delay
  00000089        00095 AVOUT    equ 0x89     ;text equate - select VOUT channel
  00000099        00096 AVUNREG  equ 0x99     ;text equate - select VUNREG channel
                  00097
                  00098
;*****************************************************************************
                  00099 ;*       Reset vector
                  00100 ;*           This is the reset vector
                  00101 ;*
                  00102
;*****************************************************************************
                  00103
0000              00104          org     0x00    ;reset vector
0000 280E         00105          goto    Main    ;program start
                  00106
                  00107
                  00108
;*****************************************************************************
                  00109 ;*       INTERRUPT SERVICE ROUTINE
```

```
                  00110 ;*           This ISR counts timer2 interrupts
                  00111 ;*
                  00112 ;*       Input Variables:
                  00113 ;*       T2POST       Counts overflow interrupts
                  00114 ;*
                  00115 ;*   Output Variables:
                  00116 ;*       T2POST       Counts overflow interrupts
                  00117 ;*
                  00118
;*************************************************************************
                  00119
0004              00120       org     0x04      ;interrupt service routine
                  00121
0004 00B7         00122       movwf   ISRW      ;save W
0005 0E03         00123       swapf   STATUS,W  ;get status
0006 00B6         00124       movwf   ISRS      ;save status
                  00125
0007 108C         00126       bcf     PIR1,TMR2IF;clear interrupt request flag
                  00127
0008 0AB5         00128       incf    T2POST,F  ;increment interrupt counter
                  00129
0009 0E36         00130       swapf   ISRS,W    ;get status
000A 0083         00131       movwf   STATUS    ;restore status
000B 0EB7         00132       swapf   ISRW,F    ;restore W
000C 0E37         00133       swapf   ISRW,W    ;restore W
                  00134
000D 0009         00135       retfie            ;return from interrupt
                  00136
                  00137
;*************************************************************************
                  00138 ;*   Main    Beginning of main loop
                  00139 ;*           First segment of code initializes controller
                  00140
;*************************************************************************
                  00141
000E              00142 Main
000E 1303         00143       bcf     STATUS,RP1;bank1 initialization
000F 1683         00144       bsf     STATUS,RP0;bank1
                  00145 ;adc
0010 3004         00146       movlw   0x04      ;RA0,1,3 analog, RA2,5 digital, Vref=Vdd
0011 009F         00147       movwf   ADCON1
                  00148
0012 302F         00149       movlw   0x2f      ;RA0,1,3 analog in; RA2,5 digital in, RA4 dig out
0013 0085         00150       movwf   TRISA
                  00151 ;PWM
0014 30FF         00152       movlw   0xFF      ;set portC to all inputs
0015 0087         00153       movwf   TRISC
0016 1107         00154       bcf     TRISC,2   ;make RC2/PWM output
                  00155
0017 303F         00156       movlw   .63       ;PWM period =  78.125KHz (8 bit resolution)
0018 0092         00157       movwf   PR2
                  00158
                  00159 ;timer2
0019 148C         00160       bsf     PIE1,TMR2IE;enable timer2 postscaler interrupts
                  00161
                  00162 ;display
001A 0186         00163       clrf    TRISB     ;make PORTB all outputs
                  00164
001B 1283         00165       bcf     STATUS,RP0;select bank0
                  00166 ;adc
001C 3089         00167       movlw   AVOUT     ;(10MHz osc)set A/D conv clock(Fosc/32),
001D 009F         00168       movwf   ADCON0    ;select RA1(AN1), turn on A/D
                  00169 ;PWM
001E 1217         00170       bcf     CCP1CON,4;clear ls bits of PWM duty cycle
001F 1297         00171       bcf     CCP1CON,5
0020 0195         00172       clrf    CCPR1L    ;set PWM to 0 (turn off PWM)
```

```
                00173
0021 3004       00174           movlw   0x04      ;enable timer2, set prescale=1:1,
                                                        postscale=1:1
0022 0092       00175           movwf   T2CON
0023 300C       00176           movlw   0x0C      ;set CCP1 to PWM mode
0024 0097       00177           movwf   CCP1CON
                00178
                00179 ;*************************************************************************
                00180 ;*       Restart Clears memory
                00181 ;*
                00182 ;*           Initializes display LEDs, desired output voltage,
                00183 ;*           debounce counters
                00184 ;*
                00185 ;*           Enables interrupts
                00186 ;*
                00187 ;*       Output Variables:
                00188 ;*           SETPOINT    desired output voltage, set to 3.0V
                00189 ;*           DNCL, DNCH  Down voltage button debounce counter
                00190 ;*           UPCL, UPCH  Up voltage button debounce counter
                00191 ;*
                00192 ;*************************************************************************
                00193
0025 3020       00194 Restart   movlw   0x20      ;clear memory from 0x20 to 0x3f
0026 0084       00195           movwf   FSR
0027 0180       00196 ClrMem    clrf    INDF
0028 0A84       00197           incf    FSR,F
0029 1F04       00198           btfss   FSR,6
002A 2827       00199           goto    ClrMem
                00200
002B 3001       00201           movlw   0x01      ;light 3.0V LED
002C 0086       00202           movwf   PORTB
                00203
002D 304D       00204           movlw   0x4d      ;initial 3.0V setting
002E 00A4       00205           movwf   SETPOINT
                00206
002F 01A2       00207           clrf    DNCL      ;clear down button debounce counter
0030 01A0       00208           clrf    UPCL      ;clear up button debounce counter
                00209
0031 30F9       00210           movlw   DEL1
0032 00A3       00211           movwf   DNCH      ;preset down button debounce counter byte
0033 00A1       00212           movwf   UPCH      ;preset up button debounce counter high byte
                00213
0034 170B       00214           bsf     INTCON,PEIE ;enable peripheral interrupt sources
0035 178B       00215           bsf     INTCON,GIE  ;enable all interrupts
                00216
                00217 ;*************************************************************************
                00218 ;*
                00219 ;*       Again   Top of main loop
                00220 ;*
                00221 ;*               Waits for 8 timer2 interrupts
                00222 ;*
                00223 ;*               A/D converts VOUT
                00224 ;*
                00225 ;*               Acquires VUNREG channel
                00226 ;*
                00227 ;*       Input Variables:
                00228 ;*               T2POST      Timer2 interrupt counter
                00229 ;*
                00230 ;*       Output Variables:
                00231 ;*               VOUT        Conversion result
                00232 ;*
                00233 ;*************************************************************************
                00234
0036            00235 Again
0036 1DB5       00236           btfss   T2POST,3    ;long delay
0037 2836       00237           goto    Again       ;try again
```

```
                00238
0038 01B5       00239           clrf    T2POST      ;clear counter
                00240
                00241 ;--- start conversion - feedback
0039 151F       00242       bsf     ADCON0,GO_DONE ;start conversion
003A 0000       00243       nop
003B 191F       00244 Wc2   btfsc   ADCON0,GO_DONE ;test if done
003C 283B       00245       goto    Wc2             ;no, wait some more
003D 081E       00246       movf    ADRES,W         ;get conversion result
003E 00A5       00247       movwf   VOUT            ;save result
                00248 ;--- end conversion
003F 3099       00249       movlw   AVUNREG         ;select VUNREG channel
0040 009F       00250       movwf   ADCON0
                00251
                00252
;*************************************************************************
                00253 ;*    FErr    Finds difference (error) between SETPOINT and VOUT
                00254 ;*    E0H:E0 = SETPOINT - VOUT
                00255 ;*
                00256 ;*    Input Variables:
                00257 ;*            SETPOINT    Desired output voltage
                00258 ;*
                00259 ;*    Output Variables:
                00260 ;*            E0H:E0      Signed error
                00261
;*************************************************************************
                00262
0041            00263 FErr
0041 01B2       00264       clrf    E0H             ;clear error high byte
                00265
0042 0825       00266       movf    VOUT,W
0043 0224       00267       subwf   SETPOINT,W      ;f-w=d
0044 00B1       00268       movwf   E0              ;save new error
                00269
0045 1C03       00270       btfss   STATUS,C        ;was there a borrow?
0046 09B2       00271       comf    E0H,F           ;yes
                00272
                00273
;*************************************************************************
                00274 ;*    Ppp     Produces proportional term by multiplying E0H:E0 by Kp
                00275 ;*
                00276 ;*            PROH:PRO = E0H:E0 * Kp
                00277 ;*            Kp = 2^3 - Produced by 3 left shifts
                00278 ;*
                00279 ;*            This term forces the output close to the desired output
                00280 ;*            quickly, but will never completely eliminate the error.
                00281 ;*
                00282 ;*    Input Variables:
                00283 ;*            E0H:E0      Error found at top of loop
                00284 ;*            Kp          Proportional gain factor (constant)
                00285 ;*
                00286 ;*    Output Variables:
                00287 ;*            PROH:PRO    Proportional component
                00288 ;*
                00289
;*************************************************************************
                00290
0047            00291 Ppp
0047 0831       00292       movf    E0,W            ;move E0 to temp space
0048 00A7       00293       movwf   TEMPA
0049 0832       00294       movf    E0H,W
004A 00A8       00295       movwf   TEMPB
                00296
004B 1003       00297       bcf     STATUS,C        ;mult E0 by 2
004C 0DA7       00298       rlf     TEMPA,F
004D 0DA8       00299       rlf     TEMPB,F
```

```
                  00300
004E 1003         00301       bcf     STATUS,C        ;mult E0 by 2
004F 0DA7         00302       rlf     TEMPA,F
0050 0DA8         00303       rlf     TEMPB,F
                  00304
0051 1003         00305       bcf     STATUS,C        ;mult E0 by 2
0052 0DA7         00306       rlf     TEMPA,F
0053 0DA8         00307       rlf     TEMPB,F
                  00308
0054 0827         00309 PppD  movf    TEMPA,W         ;move result in temp space to pro
0055 00AB         00310       movwf   PRO
0056 0828         00311       movf    TEMPB,W
0057 00AC         00312       movwf   PROH
                  00313
                  00314
;*****************************************************************************
                  00315 ;*    DifCom  Computes differential component
                  00316 ;*
                  00317 ;*        Finds difference between this loop error and previous
                                  loop error
                  00318 ;*    DIFH:DIF = (E1H:E1 - E0H:E0) * Kd
                  00319 ;*
                  00320 ;*        Kd = 2^3 - Produced by 3 left shifts
                  00322 ;*        This term tends to slow controller response.
                  00323 ;*
                  00324 ;*    Input Variables:
                  00325 ;*        E1H:E1      Previous loop error
                  00326 ;*        E0H:E0      Present loop error
                  00327 ;*        Kd          Differential gain factor (constant)
                  00328 ;*
                  00329 ;*    Output Variables:
                  00330 ;*        DIFH:DIF    differential component
                  00331 ;*
                  00332
;*****************************************************************************
                  00333
0058              00334 DifCom
0058 0834         00335       movf    E1H,W           ;get prev error high byte
0059 0232         00336       subwf   E0H,W           ;f-w=d   E0-E1=w
005A 00AE         00337       movwf   DIFH            ;save difference high byte
                  00338
005B 0833         00339       movf    E1,W            ;get prev error low byte
005C 0231         00340       subwf   E0,W            ;f-w=d   E0-E1=w
005D 00AD         00341       movwf   DIF             ;save difference low byte
                  00342
005E 1C03         00343       btfss   STATUS,C        ;was there a borrow?
005F 03AE         00344       decf    DIFH,F          ;yes
                  00345
                  00346       ;allow difference to be modified here
                  00347
0060 1003         00348       bcf     STATUS,C        ;mult dif by 2
0061 0DAD         00349       rlf     DIF,F
0062 0DAE         00350       rlf     DIFH,F
                  00351
0063 1003         00352       bcf     STATUS,C        ;mult dif by 2
0064 0DAD         00353       rlf     DIF,F
0065 0DAE         00354       rlf     DIFH,F
                  00355
0066 1003         00356       bcf     STATUS,C        ;mult dif by 2
0067 0DAD         00357       rlf     DIF,F
0068 0DAE         00358       rlf     DIFH,F
                  00359
                  00360
                  00361
                  00362
;*****************************************************************************
```

```
                00363 ;*    IntCom  Computes integral component
                00364 ;*
                00365 ;*    Multiplies present error by Ki,
                00366 ;*    adds result to INTH:INT
                00367 ;*
                00368 ;*    INTH:INT = INTH:INT + E0H:E0 * Ki
                00369 ;*
                00370 ;*    Ki = 2^3 -- Produced by 3 left shifts
                00371 ;*
                00372 ;*    This term will eliminate all error,
                00373 ;*    but not quickly
                00374 ;*
                00375 ;*    Input Variables:
                00376 ;*    E0H:E0      Present loop error
                00377 ;*    INTH:INT    Running total of errors
                00378 ;*    Ki          Integral gain factor (constant)
                00379 ;*
                00380 ;*    Output Variables:
                00381 ;*    DIFH:DIF    differential component
                00382
;*****************************************************************************
                00383
0069            00384 IntCom
0069 0832       00385       movf    E0H,W       ;move E0 to temp space
006A 00A8       00386       movwf   TEMPB
006B 0831       00387       movf    E0,W
006C 00A7       00388       movwf   TEMPA
                00389
                00390       ;allow error to be modified here before adding to integral
                00391
006D 1003       00392       bcf     STATUS,C
006E 0DA7       00393       rlf     TEMPA,F     ;E0
006F 0DA8       00394       rlf     TEMPB,F     ;E0H
                00395
0070 1003       00396       bcf     STATUS,C
0071 0DA7       00397       rlf     TEMPA,F     ;E0
0072 0DA8       00398       rlf     TEMPB,F     ;E0H
                00399
0073 1003       00400       bcf     STATUS,C
0074 0DA7       00401       rlf     TEMPA,F     ;E0
0075 0DA8       00402       rlf     TEMPB,F     ;E0H
                00403
0076 0827       00404 IntD  movf    TEMPA,W     ;get current error, E0
0077 07A9       00405       addwf   INT,F       ;add to INT, store in INT
0078 1803       00406       btfsc   STATUS,C    ;was there a carry?
0079 0AAA       00407       incf    INTH,F      ;yes, inc INT high byte
                00408
007A 0828       00409       movf    TEMPB,W     ;get E0 high byte, E0H
007B 07AA       00410       addwf   INTH,F
                00411
                00412
;*****************************************************************************
                00413 ;*    Total   Sums proportional, integral, and differential terms
                00414 ;*
                00415 ;*        PWMH:PWM = INTH:INT + PROH:PRO + DIFH:DIF
                00416 ;*
                00417 ;*        If the result should ever go negative,
                00418 ;*        the result is forced to 0,and the overload LED is set.
                00419 ;*        (This is an error condition.Can't have a negative PWM.)
                00420 ;*
                00421 ;*    Input Variables:
                00422 ;*        INTH:INT    Integral term
                00423 ;*        PROH:PRO    Proportional term
                00424 ;*        DIFH:DIF    Differential term
                00425 ;*
                00426 ;*    Output Variables:
```

```
                00427 ;*       PWMH:PWM    Sum of terms
                00428
;*****************************************************************************
                00429
007C            00430 Total
007C 082C       00431 PCom     movf    PROH,W      ;add in proportional term
007D 00B0       00432          movwf   PWMH
                00433
007E 082B       00434          movf    PRO,W
007F 00AF       00435          movwf   PWM
                00436
0080 082A       00437 ICom     movf    INTH,W      ;add in integral term
0081 07B0       00438          addwf   PWMH,F
                00439
0082 0829       00440          movf    INT,W
0083 07AF       00441          addwf   PWM,F
0084 1803       00442          btfsc   STATUS,C
0085 0AB0       00443          incf    PWMH,F
                00444
0086 082E       00445 DCom     movf    DIFH,W      ;add in differential term
0087 07B0       00446          addwf   PWMH,F
                00447
0088 082D       00448          movf    DIF,W
0089 07AF       00449          addwf   PWM,F
008A 1803       00450          btfsc   STATUS,C
008B 0AB0       00451          incf    PWMH,F
                00452
008C 1BB0       00453 Ovrld    btfsc   PWMH,7      ;did PWM go negative?
008D 2890       00454          goto    NegPwm      ;yes
008E 1306       00455          bcf     PORTB,6     ;no - turn off overload LED
008F 2893       00456          goto    PwmGen
                00457
0090 1706       00458 NegPwm   bsf     PORTB,6     ;turn on overload LED
0091 01B0       00459          clrf    PWMH        ;set PWM to 0
0092 01AF       00460          clrf    PWM
                00461
                00462
;*****************************************************************************
                00463 ;*       PwmGen  Divides PWHM:PWM by 8 (3 right shifts)
                00464 ;*           2 LSbits of PWM sent to 2 LSbits of duty cycle register
                00465 ;*           remaining 6 bits sent to CCPR1L (duty cycle register)
                00466 ;*
                00467 ;*           A/D has been acquiring VUNREG, start conversion.
                00468 ;*
                00469 ;*       Input Variables:
                00470 ;*               PWMH:PWM    PWM drive
                00471
;*****************************************************************************
                00472
009             00473 PwmGen
0093 0CB0       00474          rrf     PWMH,F      ;PWMH
0094 0CAF       00475          rrf     PWM,F       ;PWM
                00476
0095 0CB0       00477          rrf     PWMH,F      ;PWMH
0096 0CAF       00478          rrf     PWM,F       ;PWM
                00479
0097 0CB0       00480          rrf     PWMH,F      ;PWMH - can ignore contents of PWMH now
0098 0CAF       00481          rrf     PWM,F       ;PWM
                00482
0099 1217       00483          bcf     CCP1CON,4   ;clear ls bits of PWM duty cycle
009A 1297       00484          bcf     CCP1CON,5
                00485
009B 0CAF       00486          rrf     PWM,F       ;shift carry INTo PWM, lsbit INTo carry
009C 1803       00487          btfsc   STATUS,C    ;is carry set?
009D 1617       00488          bsf     CCP1CON,4   ;set PWM duty cycle lsb
                00489
```

```
009E 0CAF      00490          rrf     PWM,F       ;shift carry INTo PWM, lsbit INTo carry
009F 1803      00491          btfsc   STATUS,C    ;is carry set?
00A0 1697      00492          bsf     CCP1CON,5   ;set PWM duty cycle lsb
               00493
00A1 082F      00494      movf    PWM,W           ;get PWM
00A2 393F      00495      andlw   0x3f            ;mask off 2 ms bits (78.125KHz)
00A3 0095      00496      movwf   CCPR1L          ;put in PWM duty cycle
               00497
00A4 151F      00498      bsf     ADCON0,2        ;start conversion VUNREG
               00499
               00500
;*******************************************************************************
               00501 ;*   up/dn buttons
               00502 ;*           Debounces UP and DOWN buttons
               00503 ;*               Increments counters once for each pass
               00504 ;*               through the loop when button pressed.
               00505 ;*
               00506 ;*               If button not pressed, counter is reset.
               00507 ;*
               00508 ;*               If a button is successfully debounced,
               00509 ;*               both counters are reset.
               00510 ;*
               00511 ;*               Debounce delay is about 0.5 seconds
               00512 ;*
               00513 ;*           Moves voltage indicator bit as required.
               00514 ;*
               00515 ;*           Finds index into lookup table, and calls table.
               00516 ;*
               00517 ;*           Saves result from lookup table as new SETPOINT
               00518 ;*
               00519 ;*   Input Variables:
               00520 ;*           PORTB         Current indicator data
               00521 ;*           DNCH:DNCL     Down button debounce counter
               00522 ;*           UPCH:UPCL     Up button debounce counter
               00523 ;*
               00524 ;*   Output Variables:
               00525 ;*           PORTB         Current indicator data
               00526 ;*           DNCH:DNCL     Down button debounce counter
               00527 ;*           UPCH:UPCL     Up button debounce counter
               00528 ;*           SETPOINT      New voltage setpoint
               00529
;*******************************************************************************
               00530
00A5 0806      00531      movf    PORTB,W      ;move LED data to temp space
00A6 393F      00532      andlw   0x3f         ;mask off non-voltage LEDs
00A7 00A7      00533      movwf   TEMPA
               00534
00A8 1E85      00535 Dnb  btfss   PORTA,5      ;down
00A9 28B3      00536      goto    Upb          ;down not pushed
00AA 0FA2      00537      incfsz  DNCL,f       ;down is pushed, inc debounce
00AB 28CE      00538      goto    Wc3          ;no carry - go to next module
00AC 0FA3      00539      incfsz  DNCH,f       ;inc debounce counter high byte
00AD 28CE      00540      goto    Wc3          ;no carry - go to next module
               00541                           ;---------------------- select next lower LED
00AE 1003      00542      bcf     STATUS,C     ;select next lower LED
00AF 0CA7      00543      rrf     TEMPA,F      ;shift LED data down 1 voltage
00B0 1803      00544      btfsc   STATUS,C     ;3V LED was set before rotate, so
00B1 1427      00545      bsf     TEMPA,0      ;set it again
               00546
00B2 28BE      00547      goto    Dunb
               00548
00B3 1D05      00549 Upb  btfss   PORTA,2      ;up button
00B4 28C9      00550      goto    Nob          ;up not pushed - no buttons pushed
00B5 0FA0      00551      incfsz  UPCL,f       ;down is pushed, inc debounce
00B6 28CE      00552      goto    Wc3          ;no carry - go to next module
00B7 0FA1      00553      incfsz  UPCH,F       ;inc debounce counter high byte
```

```
00B8 28CE      00554       goto    Wc3          ;no carry - go to next module
               00555                            ;-------------------- select next higher LED
00B9 1003      00556       bcf     STATUS,C     ;select next higher voltage LED
00BA 0DA7      00557       rlf     TEMPA,F      ;shift LED data up 1 voltage
00BB 1B27      00558       btfsc   TEMPA,6      ;if 9V LED was set before,
00BC 16A7      00559       bsf     TEMPA,5      ;set it again, and
00BD 1327      00560       bcf     TEMPA,6      ;clear the overload LED
               00561
00BE 0827      00562 Dunb  movf    TEMPA,W      ;move LED data back to PORTB
00BF 0086      00563       movwf   PORTB
               00564
00C0 01A8      00565       clrf    TEMPB
00C1 03A8      00566       decf    TEMPB,F      ;set TEMPB to -1
               00567
00C2 0AA8      00568 NewSetincf    TEMPB,F      ;count up
00C3 0CA7      00569       rrf     TEMPA,F      ;rotate least sig bit INTo carry
00C4 1C03      00570       btfss   STATUS,C     ;is carry set now?
00C5 28C2      00571       goto    NewSet       ;no, try again
               00572
00C6 0828      00573       movf    TEMPB,W      ;yes, put count in w
00C7 20DD      00574       call    Tbl          ;get corresponding value for PWM
00C8 00A4      00575       movwf   SETPOINT     ;put value in SETPOINT
               00576
00C9 01A2      00577 Nob   clrf    DNCL         ;clear down button debounce counter
00CA 01A0      00578       clrf    UPCL         ;clear up button debounce counter
00CB 30F9      00579       movlw   DEL1
00CC 00A3      00580       movwf   DNCH         ;preset down button debounce counter
00CD 00A1      00581       movwf   UPCH         ;preset up button debounce counter high byte
               00582
               00583
;*****************************************************************************
               00584 ;*    Wc3     VUNREG has been converted by now,
               00585 ;*            Get result, save in VUNREG.
               00586 ;*
               00587 ;*            Select VOUT to aquire.
               00588 ;*
               00589 ;*            Test VUNREG to see if it has dropped too low.
               00590 ;*            If too low, call protective shut-down.
               00591 ;*
               00592 ;*    Input Variables:
               00593 ;*            VUNREG       Input voltage.
               00594
;*****************************************************************************
               00595
00CE           00596 Wc3
00CE 191F      00597       btfsc   ADCON0,GO_DONE ;test if done
00CF 28CE      00598       goto    Wc3            ;no, wait some more
00D0 081E      00599       movf    ADRES,W        ;get conversion result
00D1 00A6      00600       movwf   VUNREG         ;save result
               00601
00D2 3089      00602       movlw   AVOUT          ;select feedback channel to aquire
00D3 009F      00603       movwf   ADCON0
               00604
00D4 0826      00605       movf    VUNREG,W       ;get UNREG
00D5 3C50      00606       sublw   0x50           ;10V-VUNREG=?  C=1 if VUNREG<10V
00D6 1803      00607       btfsc   STATUS,C
00D7 28E4      00608       goto    ShutDn         ;turn off regulator (dies here)
               00609
               00610
               00611
;*****************************************************************************
               00612 ;*    Shift   shift errors - save current error as old
               00613 ;*            E1H:E1 = E0H:E0
               00614 ;*
               00615 ;*            Go back to top of loop.
               00616 ;*
```

```
                00617 ;*    Input Variables:
                00618 ;*            E0H:E0          Present error
                00619 ;*
                00620 ;*    Output Variables:
                00621 ;*            E1H:E1          Previous error
                00622
;*****************************************************************************
                00623
00D8            00624 Shift
00D8 0831       00625       movf    E0,W
00D9 00B3       00626       movwf   E1
                00627
00DA 0832       00628       movf    E0H,W
00DB 00B4       00629       movwf   E1H
                00630
00DC 2836       00631       goto    Again
                00632
                00633
;*****************************************************************************
                00634 ;*    Tbl     Look up table.
                00635 ;*
                00636 ;*            Called with an index in W.
                00637 ;*
                00638 ;*            Index is added to program counter.
                00639 ;*
                00640 ;*            Execution jumps to "retlw" which will return
                00641 ;*            to the calling program with table value in w.
                00642 ;*
                00643 ;*    Input Variables:
                00644 ;*            w               Offset index into table
                00645 ;*
                00646 ;*    Output Variables:
                00647 ;*            w               Value from table
                00648
;*****************************************************************************
                00649
00DD            00650 Tbl                           ;call with index in w
00DD 0782       00651       addwf   PCL,F           ;add index to PC
                00652
00DE 344D 3474 3482 00653           dt      0x4d, 0x74, 0x82, 0x9b, 0xc2, 0xea
     349B 34C2 34EA
                00654 ;output voltage               3.0   4.5   5.0   6.0   7.5   9.0
                00655
                00656
;*****************************************************************************
                00657 ;*    ShutDn  Protective shutdown. Entry into this routine is a result
                00658 ;*    of a low input voltage. This turns off the converter
                00659 ;*    before the series pass transistor can overheat.
                00660 ;*
                00661 ;*            PWM on-time is set to 0, turning off the converter.
                00662 ;*
                00663 ;*            The converter trip LED is lit, indicating shutdown.
                00664 ;*
                00665 ;*            Execution then loops without exit. The only exit is to
                00666 ;*            reset the controller.
                00667
;*****************************************************************************
                00668
00E4            00669 ShutDn
00E4 1217       00670       bcf     CCP1CON,4       ;turn off PWM
00E5 1297       00671       bcf     CCP1CON,5
00E6 0195       00672       clrf    CCPR1L
                00673
00E7 1786       00674       bsf     PORTB,7         ;light trip LED
                00675
00E8            00676 Dead
00E8 28E8       00677       goto    Dead            ;stays here
```

```
           00678
           00679
           00680             END


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXX------- ----------------
2000 : -------X-------- ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   230
Program Memory Words Free:  1818


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     7 suppressed
```

RS-232 Autobaud for the PIC16C5X Devices

Author: *Thomas Schmidt*
Microchip Technology

INTRODUCTION

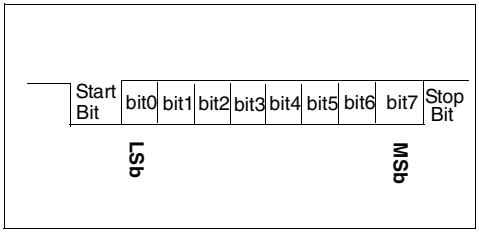
This application note describes an implementation of a RS-232 Autobaud routine on a PIC16C54B microcontroller.

Many microcontroller applications require chip-to-chip serial communication. Since the PIC16C54B has no USART, serial communication must be performed in software. Some applications use multiple transmission rates. Multiple transmission rates require software which detects the transmission rate and adjusts the receive and transmit routines according to the transmission rate.

ASYNCHRONOUS SERIAL I/O COMMUNICATION

Figure 1 shows the format of a data byte transferred via a serial communication line. Before the actual data byte is going to be transmitted, the data line is set to a high level. The first bit transmitted is called the start-bit and is always low, followed by the actual data. The data is transmitted with the LSb (last-significant-bit) first and the MSb (most significant bit) last. A high level represents a one bit and a low level a zero bit. The final bit transmitted is the stop-bit. The stop-bit is always a logic high.

FIGURE 1: DATA BYTE



The number of bits transmitted per second is equal to the baud rate. The inverse of the baud rate equals to the transmission time for one bit.

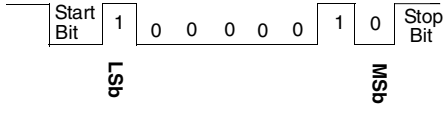
EXAMPLE 1: BAUDRATE CALCULATION

$$t_{one-bit} = \frac{1}{9600 Baud} = 104\mu s$$

In asynchronous communication, the receiver must know the baud rate of the transmitter, because only the data shown in Figure 1 is transmitted. No clock is provided by the transmitter.

Example 2 depicts the asynchronous transmission of the character 'A'. The character 'A' has the value 41_h (ASCII).

EXAMPLE 2: ASYNCHRONOUS TRANSMISSION OF CHARACTER 'A'



Note: Character 'A' is equivalent to 41_H . 41_H=01000001_b

Autobaud and Asynchronous Serial Communication

In some systems, the transmission is not fixed to a baud rate. In this case, the received has to adjust the baud rate to that of the transmitter. Autobaud means that the receiver measures the transmission time of a calibration character and adjusts the delay routines for the baud rate generation accordingly.

THE SYSTEM

This chapter gives an overview on the setup of the hardware and software.

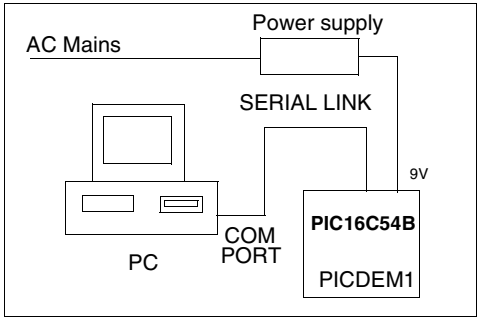
The Hardware

In this application, a PIC16C54B is connected to a PC. The PIC16C54B is placed on a PICDEM1 board. The PICDEM1 board provides a DSUB9 connector to a PC and a MAX232 interface circuit.

The PICDEM1 board is connected via the DSUB9 connector and a serial cable to the serial port of the PC. In this application, the PC sends a calibration character to the PIC16C54B. The PIC16C54B detects the transmission rate by measuring the bit length of transmitted zeros in a calibration character. The transmission time is measure by a software counter. The value of the software counter represents the value of the transmission rate for one bit. This value is used to generate a delay for bit sampling.

The hardware setup for this application note is shown in Figure 2.

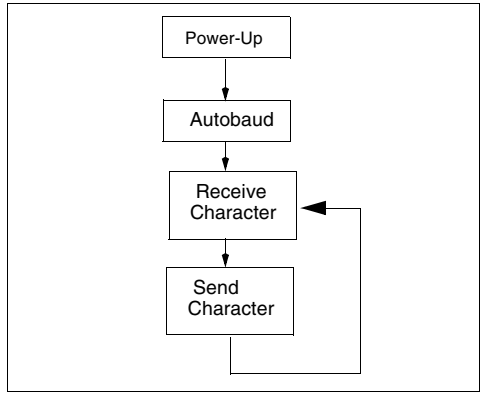
FIGURE 2: HARDWARE SETUP



The Program Flow

The program flow is shown in Figure 3.

FIGURE 3: PROGRAM FLOW OF THE MAIN ROUTINE



After power-up, the PIC16C54B initializes the I/O ports and waits for a calibration character from the PC. When the PC sends the calibration character, the PIC16C54B measures the transmission rate. This is done within the autobaud routine. Once the transmission rate has been detected, the PC has to send a second character. This character is received and echoed to the PC by the PIC16C54B. This process, receiving and transmitting characters, runs in an infinite loop.

The software is divided into three modular routines:

- Autobaud routine
- Receive routine
- Transmit routine

Each routine is a separate software module and can easily be integrated in custom code.

The communication between the PC and the PIC16C54B is half-duplex. In order to implement a full-duplex communication, please refer to *AN510 Implementation Of An Asynchronous Serial I/O*.

THE AUTOBAUD ROUTINE

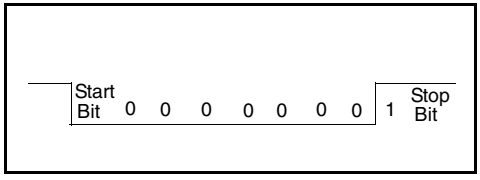
This chapter describes the theory of operation and the implementation of the autobaud routine.

Note: The software is designed for a 8-N-1 communication. Where 8 equals the number of data bits (start and stop bit not included), N is equal to the no parity bit and 1 is equal to the one stop bit.

In order to adjust to the transmission rate on the receiver side, the transmitter has to send a known character to the receiver. This character is called the calibration character. The receiver must know the pattern of the character, so it can measure the time to receive one or more bits. From the measured time, the receiver calculates the transmission time for one bit. This time is used in a receive or transmit routine to generate the baud rate.

The calibration value used for the autobaud routine in this application note is shown in Figure 4.

FIGURE 4: CALIBRATION CHARACTER FOR AUTOBAUD ROUTINE



In the first step, the autobaud routine looks for the start-bit. After the start-bit has been detected, a 16-bit software counter will increment until the next low to high transition is detected (see Figure 4). This means the autobaud routine measures the transmission time of eight zeros (including the start-bit).

The value in the counter represents the value for the transmission rate for 8 zeros. In order to calculate the transmission time for one bit, the value of the 16-bit counter is divided by 8. The result is the transmission time for one bit.

While measuring the transmission time and calculating the transmission time for one bit, the autobaud routine has to check if the 16-bit counter overflows or the result of the division could be zero. A counter overflow means that the transmitted signal is to slow. If the division by 8 equals zero, that means that the incoming signal is too fast.

The Implementation

The implementation of the autobaud routine can be broken up into 6 sections.

1. Check for start-bit
2. Measure time (increment counter)
3. Divide measured time by eight
4. Calculate time for half the transmission time for one bit (divide previous result by two). Half the baudrate is used in the receive routine to place the sampling of the bits in the middle.
5. Adjust result for receive and transmit routines
6. Check if both calculated results are greater than zero. If one of the results is zero, the baudrate cannot be generated because the received signal was to fast.

Each of this sections will be explained separately in the following text. The entire source code for the autobaud, as well as the receive and transmit routines, are given in the Appendix.

Check for Start-Bit

In the first step, the autobaud routine is called and the registers are initialized (see Figure 5). The low and the high byte of the autobaud counter are set to zero. The autobaud status register is also cleared. The autobaud status register contains two error flags, which indicate if the incoming signal was too fast or too slow. After the initialization, the receive pin RX is checked for a high to low transition. When this is detected, the autobaud routine starts measuring.

FIGURE 5: CHECK FOR START BIT

```
Autobaud        clrf    AUTOBAUD_LOW        ; reset register
                clrf    AUTOBAUD_HIGH       ; reset register
                clrf    AUTOHALF_LOW        ; reset register
                clrf    AUTOHALF_HIGH       ; reset register
                clrf    AUTOB_STATUS        ; reset autobaud
                                            ; status register
TestStartBit
                btfsc   PORTA, RX           ; check for start-bit
                goto    TestStartBit        ; Start-bit not found
```

Measure Time To Receive Calibration Word

After the start-bit is detected, the autobaud routine measures the time to receive the calibration character. The source code of this section is shown in Figure 6. The calibration character has the pattern 10000000b. The autobaud routine increments a 16-bit counter until a low to high transition is found. The registers for the 16-bit counter are called AUTOBAUD_HIGH (high byte) and AUTOBAUD_LOW (low byte). If the high byte overflows the error flag SIGNAL_SLOW in the register, AUTOBAUD_STATUS will be set. An overflow means that the incoming signal is too slow, because it takes more cycles to increment the counter than to transmit the full calibration character. See Figure 6.

Calculate Transmission Time For One Bit

After all bits are received the measured time has to be divided by eight, because the time to receive eight zeros was measured. The division is simply done by shifting the 16-bit counter three times to the right. Zeros are shifted into the counter from the left side. The transmission time for one bit is stored in the registers AUTOBAUD_LOW and AUTOBAUD_HIGH.

FIGURE 6: MEASURE TIME TO RECEIVE CALIBRATION WORD

```
Autobaud       clrf    AUTOBAUD_LOW       ; reset register
TestBitHigh    btfsc   PORTA, RX          ; Test for end of bit stream
               goto    Calculate          ; End of bit stream, now calculate
                                          ; bit time for one bit
               incfsz  AUTOBAUD_LOW, f    ; increment Autobaud low register
               goto    TestBitHigh        ; test for high bit
               incfsz  AUTOBAUD_HIGH, f   ; increment high byte of autobaud register
               goto    TestBitHigh        ; test for end of bit stream
               goto    Signal2Slow        ; High byte got an overflow. Transmitted
                                          ; signal is to slow for clock speed of the uc
```

FIGURE 7: CALCULATION OF TRANSMISSION TIME FOR ONE BIT

```
Autobaud       clrf    AUTOBAUD_LOW       ; reset register
                                          ; divide by measure time by 8 (8 zero where transmitted
                                            including
                                          ; start-bit)
Calculate      movlw   0x03               ; Initialize count register
               movwf   COUNTER            ; Counter for number for rotates = 3
Divide         bcf     STATUS, C          ; clear carry bit
               rrf     AUTOBAUD_HIGH,f    ; rotate autobaud high register
               rrf     AUTOBAUD_LOW, f    ; rotate autobaud low register
               decfsz  COUNTER, f         ; decrement counter
               goto    Divide             ; divide
```

Calculate Half The Bit Time

After the transmission time for one bit is calculated, the transmission time for half the bit time has to be computed. This value is needed in the received routine to place sampling in the middle of each bit. After the start bit has been detected in the receive routine, the routine waits 1.5 bit times before the first data bit is sampled. This ensures that the sampling always happens in the middle of the bit.

The calculation of half the bit time is done by simply shifting the 16-bit counter to the right once. The result of the division is stored in the registers AUTOHALF_HIGH and AUTOHALF_LOW. The source code for this section of the autobaud routine is shown in Figure 8.

Adjust Transmission Times For Receive and Transmit Routine

The value of the 16-bit counter for the full bit time and the value for half the bit time have to be adjusted for the receive and transmit routine. Each count in the register AUTOBAUD_LOW and AUTOHALF_LOW stands for 5 instruction cycles, because it took five instruction cycles to get one count. Since the receive and transmit routines have a software overhead for storing or restoring data, this overhead has to be subtracted from the counter values.

After each adjustment, the result is checked to see if it is negative. If this is the case, error flag SIGNAL2FAST will be set. See Figure 9.

FIGURE 8: CALCULATION OF HALF THE BIT TIME

```
Autobaud        clrf    AUTOBAUD_LOW        ; reset register
                                            ; Calculate half the bit time
CalcHalfBit     bcf     STATUS, C           ; clear carry bit
                rrf     AUTOBAUD_HIGH,w     ; rotate autobaud high register
                movwf   AUTOHALF_HIGH       ; copy result into AUTOHALF_HIGH register
                rrf     AUTOBAUD_LOW, w     ; rotate autobaud high register
                movwf   AUTOHALF_LOW        ; copy result into AUTOHALF_LOW register
```

FIGURE 9: COUNTER ADJUSTMENT AND CHECK IF COUNTERS ARE NEGATIVE

```
Autobaud        clrf    AUTOBAUD_LOW        ; reset register
AdjustLowByte   movlw   0x3                 ; 18-19 instruction cycles overhead from
                                            ; transmit and receive routine. This overhead
                                            ; must be subtracted from iterations
                subwf   AUTOBAUD_LOW, f     ; Adjust low byte from Autobaud counter
                btfss   STATUS, C           ; Is result negative? (equal=0 will be checked
                                            ; at ErrorCheck). C=0 result is negative
                goto    Signal2Fast         ; Signal is to fast for receive and transmit routine
                movlw   0x02                ; subtract 2 from low byte of half the bit time
                subwf   AUTOHALF_LOW, f     ; subtract from low byte of half the bit time
                btfss   STATUS, C           ; Is result negative? (equal=0 will be checked
                                            ; at ErrorCheck). C=0 result is negative
                goto    Signal2Fast         ; Signal is to fast for receive and transmit routine
```

Check If Both Counter Values Are Zero

After the adjustment, both counter values for the full and half bit time are checked for zeros. If this is the case, the error flag SIGNAL2FAST is set. If both counters are greater than or equal to one, the autobaud routine returns to the main routine. The source code for this section of the autobaud routine is shown in Figure 10.

FIGURE 10: CHECK OF COUNTER VALUES

```
Autobaud        clrf    AUTOBAUD_LOW              ; reset register
                                                  ; check if AUTOBAUD_HIGH and AUTOBAUD_LOW are
                                                    zero.
                                                  ; This means the transmission time for one byte
                                                    is too high
ErrorCheck      movf    AUTOBAUD_HIGH,w           ; copy high byte of autobaud counter register
into
                                                  ; w-register
                xorwf   AUTOBAUD_LOW, w           ; AUTOBAUD_HIGH = AUTOBAUD_LOW?
                btfss   STATUS, Z                 ; is result zero?
                goto    ErrorCheckHalf            ; Result is not zero, therefore finish autobaud
                                                  ; routine
                goto    Signal2Fast               ; Signal is to fast for routine
ErrorCheckHalf  movf    AUTOHALF_HIGH,w           ; copy high byte of autobaud counter register
into
                                                  ; w-register
                xorwf   AUTOHALF_LOW, w           ; AUTOBAUD_HIGH = AUTOBAUD_LOW?
                btfss   STATUS, Z                 ; is result zero?
                goto    EndAutoBaud               ; Result is not zero, therefore finish autobaud
                                                  ; routine
                                                  ; Error: delay for half the bit time is zero,
                                                    therefore a
                                                  ; delay cannot be generated with the delay
                                                    routines. Incoming signal
                                                  ; is to fast for clock speed.
Signal2Fast     bsf     AUTOB_STATUS, SIGNAL_FAST ; set error flag
                retlw   0x00                      ; return to operating system
Signal2Slow     bsf     AUTOB_STATUS, SIGNAL_SLOW ; set error flag
EndAutoBaud     retlw   0x00                      ; Return to operating system
```

THE TRANSMIT ROUTINE

The source code for the transmit routine is shown in Figure 11.

FIGURE 11: SOURCE CODE OF THE TRANSMIT ROUTINE

```
                                                  ; Transmit routine
                                                  ; Transmits LSB first
                                                  ; Software overhead = 10 instruction cycles
                                                    (including call
                                                  ; to DelayFullBit routine, return from
                                                  ; delay routine not included)
Transmit        movlw       BITS                  ; Number of bit's to transmit
                movwf       COUNTER               ; Initialize count register
                bcf         PORTA, TX             ; Generate start-bit
                call        DelayFullBit          ; Generate Delay for one bit-time
TransmitNext    rrf         RXTX_REG, f           ; Rotate receive register
                btfsc       STATUS, C             ; Test bit to be transmitted
                bsf         PORTA, TX             ; Transmit one
                btfss       STATUS, C             ; Check carry bit if set
                bcf         PORTA, TX             ; Transmit a zero
                call        DelayFullBit          ; call Delay routine
                decfsz      COUNTER, f            ; Decrement count register
                goto        TransmitNext          ; Transmit next bit
```

```
bsf         PORTA, TX           ; Generate Stop bit
call        DelayFullBit        ; Delay for Stop bit
retlw       0x00                ; Return to operating system
                                :
```

In the first step, the transmit routine initializes the register Count to 8. After the initialization, the RXTX_REG register is rotated by one position to the right. The bit-0 of the RXTX_Reg is now stored in the carry flag. The carry bit is checked whether it is a '1' or a '0'. If the carry bit is set, the TX-pin is also set, otherwise the TX-pin is cleared. After all bits are transmitted, the stop-bit is send.

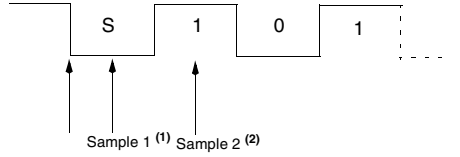
The delay for the transmission is generated by the DELAYFULLBit routine.

THE RECEIVE ROUTINE

The source code for the receive routine is shown in Figure 13.

The receive routine first resets the receive register to '0' and initializes the Count register with 8. After the initialization, the routine checks for the start-bit. When the start bit is detected ,the receive routine waits 1.5 times the transmission time of one bit before sampling the next bit. This ensures that the bits are sampled in the middle and not at the beginning or end of the bit (see Figure 12). The delay for half the bit time is generated by the routine DelayHalfBit. After the delay, the bit is sample and stored in the register RXTX_REG.

FIGURE 12: RECEIVE ROUTINE SAMPLING



Note 1: Delay is generated by using delay value from register AutoBaud2

Note 2: Delay is generated by using delay value from register AutoBaud

FIGURE 13: SOURCE CODE OF THE RECEIVE ROUTINE

```
                                        ; Receive Routine
                                        ; receive routine = 11 instruction cycles per
                                          iteration
                                        ; including call to DelayFullBit routine
Receive         clrf    RXTX_REG        ; Clear receive register
                movlw   BITS            ; Number of bits to receive
                movwf   COUNTER         ; Load number of bits into counter register
ReceiveStartBit btfsc   PORTA, RX       ; Test for start bit
                goto    ReceiveStartBit ; Startbit not found
                call    DelayHalfBit    ; Wait until middle of start bit
                call    DelayFullBit    ; Ignore start-bit and sample first
                                        ; data bit in the middle of the bit
ReceiveNext     btfsc   PORTA, RX       ; Is bit a zero or a one
                bsf     STATUS,C        ; bit is a one => set carry bit
                btfss   PORTA, RX       ; Is bit a one or a zero
                bcf     STATUS,C        ; bit is a zero => clear carry bit
                rrf     RXTX_REG, f     ; Rotate receive register
                call    DelayFullBit    ; Call Delay routine
                decfsz  COUNTER, f      ; decrement receive count register by one
                goto    ReceiveNext     ; Receive next bit
                retlw   0x00            ; back to operation system
                                        :
```

The time is measured by using a software timer. The software timer is started when the start-bit is detected. The start-bit is detected when a transition from high to low occurs. Once the start-bit is detected, the software timer counts until a low to high transition is detected.

THE DELAY ROUTINES

The delay routine for half the bit time and the full bit time are identical in program flow. If the high byte is zero, only the low byte will be decremented. For decrementing, the low byte is stored in a temporary register. When the low byte is zero, the delay routine returns to either the receive or transmit routine.

If the high byte is not zero, the low byte will be decremented n-times, where n is the value stored in the high byte.

OTHER POSSIBLE AUTOBAUD IMPLEMENTATIONS

There a several other methods to implement an autobaud routine. These methods are briefly described below. The implementations are not given within this application note.

Measuring The Bit Length Using A Timer

Instead of using a software counter, a timer can be used. This would require modifications in the autobaud and the receive and transmit routines. The disadvantage of this method is that one timer has to be dedicated to the autobaud routine.

Measuring The Bit Length Of The First Bit For Each Character Transmitted

This method measures the transmission time of the first bit from a transmitted character. The measured value is used to adjust the delay counter for receiving the following bits. The measurement is done for each character received. Variations in the oscillator frequency are compensated for using this method. The disadvantage of this method is that the transmitted characters need a zero to one transition in the first bit. This limits the number of characters which can be transmitted.

SOFTWARE PERFORMANCE

The performance of the autobaud routine is shown.

TABLE 1: SOFTWARE PERFORMANCE

Oscillator Frequency	Min. Baudrate	Max. Baudrate
4 MHz	110 Baud	19200 Baud
10 MHz	110 Baud	38400 Baud
20 MHz	110 Baud	57600 Baud

APPENDIX

```
MPASM 02.20.04 Intermediate  AUTO16B3.ASM   3-17-1999  11:28:13

00001 ; *************************************************************************************************
00002 ; * Title           : RS-232 Autobaud routine                                                     *
00003 ; * Author          : Thomas Schmidt                                                              *
00004 ; *                   Application Engineer for Standard Microcontroller and ASSP Products       *
00005 ; * Date            : 04.01.1999                                                                  *
00006 ; * Revision        : 1.0                                                                         *
00007 ; * Last Modified   : 04.01.1999                                                                  *
00008 ; * Description     : The purpose of this program to detect automatically the Baudrate of a RS-232*
00009 ; * transmitter. The detected baudrate is used to adjust a delay routine for a transmit and     *
00010 ; * receive routine.                                                                             *
00011 ; * This program measures the transmission time of an incoming calibration character. Based on  *
00012 ; * the measured time the transmission time for one bit is calculated. This value is used in    *
00013 ; * a software delay routine to generate a delay for on bit. The delay routine is called from   *
00014 ; * a transmit and receive routine. The user is free to modify the main routine. If the user    *
00015 ; * chooses to modify the receive and transmit routine he has to modify as well the software    *
00016 ; * adjustment in the autobaud routine.                                                          *
00017 ; *************************************************************************************************
00018
00019
00020                 LIST P=16C54B, r=hex
00021
00022                 ; *****************************************************************
00023                 ; * Include files                                                 *
00024                 ; *****************************************************************
00025                 #include "P16C5X.INC"
00001         LIST
00002 ; P16C5X.INC  Standard Header File, Version 4.00    Microchip Technology, Inc.
00313         LIST
00026
00027                 ; *****************************************************************
00028                 ; * Pin definitions                                               *
00029                 ; *****************************************************************
00030                 #define RX              2               ; receive pin, connected to RA2
00031                 #define TX              3               ; transmit pin, connected to RA3
00032
00033
00034                 ; *****************************************************************

00035                 ; * Register definitions                                          *
```

```
                00036               ; *****************************************************************************
                00037               cblock  0x08
00000008        00038                       AUTOBAUD_LOW    ; low byte of bit-time counter
00000009        00039                       AUTOBAUD_HIGH   ; high byte of bit-time counter
0000000A        00040                       AUTOHALF_LOW    ; low byte of half the bit time
0000000B        00041                       AUTOHALF_HIGH   ; high byte for half the bit time
0000000C        00042                       AUTOB_STATUS    ; status byte for Autobaud routine
0000000D        00043                       TEMP1, TEMP2    ; temporary registers
0000000F        00044                       RXTX_REG        ; receive register
00000010        00045                       COUNTER         ; receive & Transmit counter register
                00046               endc
                00047
                00048               ; *****************************************************************************
                00049               ; * Bit definitions in register AUTOB_STATUS                                  *
                00050               ; *****************************************************************************
                00051               #define SIGNAL_FAST     0               ; signal-to-fast flag in AUTOB_STATUS
                00052                                                       ; byte. This bit indicates that the
                00053                                                       ; incoming signal was too fast
                00054                                                       ; AUTOB_STATUS.SIGNAL_FAST=0  Signal was OK
                00055                                                       ; AUTOB_STATUS.SIGNAL_FAST=1  Signal was too fast
                00056               #define SIGNAL_SLOW     1               ; signal-to-slow flag in AUTOB_STATUS
                00057                                                       ; byte. This bit indicates that the
                00058                                                       ; incoming signal was too slow
                00059                                                       ; AUTOB_STATUS.SIGNAL_SLOW=0  Signal was OK
                00060                                                       ; AUTOB_STATUS.SIGNAL_SLOW=1  Signal was to slow
                00061
                00062
                00063
                00064               ; *****************************************************************************
                00065               ; * Other definitions                                                         *
                00066               ; *****************************************************************************
                00067               #define BITS            8               ; number of bits to receive
                00068
                00069
                00070               ; *****************************************************************************
                00071               ; * Fuse configuration                                                        *
                00072               ; *****************************************************************************
0FFF 0FF9       00073               __CONFIG _CP_OFF&_WDT_OFF&_XT_OSC
                00074
                00075               ; *****************************************************************************
                00076               ; * Reset vector                                                              *
                00077               ; *****************************************************************************
01FF            00078               ORG 0x1FF
01FF 0A00       00079               goto    Begin
                00080
                00081               ; *****************************************************************************
```

```
                    00082                ; * Program Start                                                *
                    00083                ; ***************************************************************************
0000                00084                ORG 0x00
                    00085
                    00086                ; ***************************************************************************
                    00087                ; * Initialization                                                 *
                    00088                ; ***************************************************************************
0000 0066           00089 Begin          clrf    PORTB           ; set all latches of PORTB to '0'
0001 0040           00090                clrw                    ; reset W-Register
0002 0006           00091                tris    PORTB           ; initialize TRIS register
0003 0065           00092                clrf    PORTA           ; reset latches of PortA
0004 0CF7           00093                movlw   b'11110111'     ; R2=RX, RA3=TX
0005 0005           00094                tris    PORTA           ; initialize TRIS register for PORTA
                    00095
                    00096                ; ***************************************************************************
                    00097                ; * Main routine. The main routine detects first the transmission       *
                    00098                ; * time of the incoming calibration character. After that the          *
                    00099                ; * routine receives and transmits incoming characters.                *
                    00100                ; ***************************************************************************
0006 090F           00101                call    Autobaud        ; call Autobaud routine
0007 020C           00102                movf    AUTOB_STATUS, w ; check if an error occurred
0008 0643           00103                btfsc   STATUS, Z       ; is AUTOB_STATUS=0 (means no error occurred)
0009 0A0C           00104                goto    Main            ; goto Main
                    00105
                    00106                ; An error occurred. The incoming signal was either too fast or too slow.
                    00107                ; The autobaud status register AUTOB_STATUS is displayed on PORTB in
                    00108                ; order to indicated that an error occurred. The receive and transmit
                    00109                ; routine will not be called.
000A 0026           00110                movwf   PORTB           ; display AUTOB_STATUS on PORTB
000B 0A0B           00111 DoForever      goto    DoForever       ; an error occurred. This error is displayed on PORTB.
                    00112                                        ; Because of this error, the receive and transmit
                    00113                                        ; routine will not be called.
                    00114
                    00115                ; No error occurred. There receive and transmit characters.
000C 0952           00116 Main           call    Transmit        ; transmit received character back to transmitter
000D 0942           00117                call    Receive         ; receive next character
000E 0A0C           00118                goto    Main            ; do forever
                    00119
                    00120                ; ***************************************************************************
                    00121                ; * Autobaud routine                                                 *
                    00122                ; ***************************************************************************
000F 0068           00123 Autobaud       clrf    AUTOBAUD_LOW    ; reset register
0010 0069           00124                clrf    AUTOBAUD_HIGH   ; reset register
0011 006A           00125                clrf    AUTOHALF_LOW    ; reset register
0012 006B           00126                clrf    AUTOHALF_HIGH   ; reset register
0013 006C           00127                clrf    AUTOB_STATUS    ; reset autobaud status register
0014 0645           00128 TestStartBit   btfsc   PORTA, RX       ; check for start-bit
```

```
0015 0A14          00129                goto    TestStartBit    ; start-bit not found
                   00130
0016 0645          00131 TestBitHigh    btfsc   PORTA, RX       ; test for end of bit stream
0017 0A1D          00132                goto    Calculate       ; end of bit stream, now calculate
                   00133                                        ; bit time for one bit
0018 03E8          00134                incfsz  AUTOBAUD_LOW, f ; increment Autobaud low register
0019 0A16          00135                goto    TestBitHigh     ; test for high bit
001A 03E9          00136                incfsz  AUTOBAUD_HIGH,f ; increment high byte of autobaud register
001B 0A16          00137                goto    TestBitHigh     ; test for end of bit stream
001C 0A40          00138                goto    Signal2Slow     ; high byte got an overflow. Transmitted
                   00139                                        ; signal is too slow for clock speed
                   00140
                   00141                ; Calculation of transmission time for one bit
001D 0C03          00142 Calculate      movlw   0x03            ; initialize count register
001E 0030          00143                movwf   COUNTER         ; counter for number for rotates = 3
001F 0403          00144 Divide         bcf     STATUS, C       ; clear carry bit
0020 0329          00145                rrf     AUTOBAUD_HIGH,f ; rotate autobaud high register
0021 0328          00146                rrf     AUTOBAUD_LOW, f ; rotate autobaud low register
0022 02F0          00147                decfsz  COUNTER, f      ; decrement counter
0023 0A1F          00148                goto    Divide          ; divide
                   00149
                   00150
                   00151                ; Calculate the transmission time for half the bit time (means
                   00152                ; divide transmission time of one bit by two).
0024 0403          00153 CalcHalfBit    bcf     STATUS, C       ; clear carry bit
0025 0309          00154                rrf     AUTOBAUD_HIGH,w ; rotate autobaud high register
0026 002B          00155                movwf   AUTOHALF_HIGH   ; copy result into AUTOHALF_HIGH register
0027 0308          00156                rrf     AUTOBAUD_LOW, w ; rotate autobaud high register
0028 002A          00157                movwf   AUTOHALF_LOW    ; copy result into AUTOHALF_LOW register
                   00158
                   00159                ; Adjust 16-bit counter for receive and transmit routine. This means
                   00160                ; that the overhead of instruction cycles in of the receive/transmit
                   00161                ; routine has to be subtracted from the transmission time of one bit
                   00162                ; and half a bit.
0029 0C03          00163 AdjustLowByte  movlw   0x3             ; 18-19 instruction cycles overhead from
                   00164                                        ; transmit/receive routine. This overhead
                   00165                                        ; must be subtracted from iterations
002A 00A8          00166                subwf   AUTOBAUD_LOW, f ; adjust low byte from Autobaud counter
002B 0703          00167                btfss   STATUS, C       ; is result negative? (equal=0 will be checked
                   00168                                        ; at ErrorCheck). C=0 result is negative
002C 0A3E          00169                goto    Signal2Fast     ; signal is too fast for receive and transmit routine
002D 0C02          00170                movlw   0x02            ; subtract 2 from low byte of half the bit time
002E 00AA          00171                subwf   AUTOHALF_LOW, f ; subtract from low byte of half the bit time
002F 0703          00172                btfss   STATUS, C       ; is result negative? (equal=0 will be checked
                   00173                                        ; at ErrorCheck). C=0 result is negative
0030 0A3E          00174                goto    Signal2Fast     ; signal is too fast
                   00175
```

```
                    00176                ; check if AUTOBAUD_HIGH and AUTOBAUD_LOW are zero. This
                    00177                ; means the transmission time for one byte is too high
0031 0229           00178 ErrorCheck     movf    AUTOBAUD_HIGH,f ; copy high byte of autobaud counter register onto itself
0032 0743           00179                btfss   STATUS, Z       ; is zero-flag set?
0033 0A38           00180                goto    ErrorCheckHalf  ; no, therefore check next byte
0034 0228           00181                movf    AUTOBAUD_LOW, f ; copy low byte of autobaud register onto itself
0035 0743           00182                btfss   STATUS, Z       ; is zero-flag set?
0036 0A38           00183                goto    ErrorCheckHalf  ; no, low byte is not zero therefore check next byte
0037 0A3E           00184                goto    Signal2Fast     ; yes, signal is too fast. Therefore set flag
0038 022B           00185 ErrorCheckHalf movf    AUTOHALF_HIGH,f ; copy high byte of autobaud counter onto itself
0039 0743           00186                btfss   STATUS, Z       ; is zero-flag set?
003A 0A41           00187                goto    EndAutoBaud     ; finish autobaud routine
003B 022A           00188                movf    AUTOHALF_LOW, f ; check low byte
003C 0743           00189                btfss   STATUS, Z       ; is zero-flag set?
003D 0A41           00190                goto    EndAutoBaud     ; no, therefore finish autobaud routine
                    00191                                        ; yes, High and low byte of AUTOHALF register are zero
                    00192                                        ; there the incoming signal was too fast to generate a delay
                    00193                                        ; Therefore set SIGNAL_FAST flag
                    00194
                    00195                ; Error: delay for half the bit time is zero, therefore a
                    00196                ; delay cannot be generated with the delay routines. The incoming signal
                    00197                ; was too fast for clock speed.
003E 050C           00198 Signal2Fast    bsf     AUTOB_STATUS, SIGNAL_FAST       ; set error flag
003F 0800           00199                retlw   0x00                            ; return to main routine
                    00200
0040 052C           00201 Signal2Slow    bsf     AUTOB_STATUS, SIGNAL_SLOW       ; set error flag
                    00202
                    00203
0041 0800           00204 EndAutoBaud    retlw   0x00                            ; return to main routine

                    00205
                    00206
                    00207                ; *********************************************************************
                    00208                ; * Receive Routine                                                   *
                    00209                ; *********************************************************************
0042 006F           00210 Receive        clrf    RXTX_REG        ; clear receive register
0043 0C08           00211                movlw   BITS            ; number of bits to receive
0044 0030           00212                movwf   COUNTER         ; load number of bits into counter register
0045 0645           00213 ReceiveStartBit btfsc  PORTA, RX       ; test for start bit
0046 0A45           00214                goto    ReceiveStartBit ; start-bit not found
0047 0972           00215                call    DelayHalfBit    ; wait until middle of start-bit
0048 0961           00216                call    DelayFullBit    ; ignore start-bit and sample first
                    00217                                        ; data bit in the middle of the bit
0049 0645           00218 ReceiveNext    btfsc   PORTA, RX       ; is RX zero or a one?
004A 0503           00219                bsf     STATUS,C        ; bit is a one => set carry bit
004B 0745           00220                btfss   PORTA, RX       ; is RX one or a zero?
004C 0403           00221                bcf     STATUS,C        ; RX is zero => clear carry bit
```

```
004D 032F 00222                 rrf     RXTX_REG, f     ; rotate value into receive register
004E 0961 00223                 call    DelayFullBit    ; call Delay routine
004F 02F0 00224                 decfsz  COUNTER, f      ; decrement receive count register by one
0050 0A49 00225                 goto    ReceiveNext     ; receive next bit
0051 0800 00226                 retlw   0x00            ; return to main routine
                  00227
                  00228 ;*********************************************************************
                  00229 ; * Transmit routine                                                 *
                  00230 ;*********************************************************************
0052 0C08 00231 Transmit        movlw   BITS            ; number of bit's to transmit
0053 0030 00232                 movwf   COUNTER         ; initialize count register
0054 0465 00233                 bcf     PORTA, TX       ; generate start-bit
0055 0961 00234                 call    DelayFullBit    ; generate Delay for one bit-time
0056 032F 00235 TransmitNext    rrf     RXTX_REG, f     ; rotate receive register
0057 0603 00236                 btfsc   STATUS, C       ; test bit to be transmitted
0058 0565 00237                 bsf     PORTA, TX       ; transmit a one
0059 0703 00238                 btfss   STATUS, C       ; check carry bit if set
005A 0465 00239                 bcf     PORTA, TX       ; transmit a zero
005B 0961 00240                 call    DelayFullBit    ; call Delay routine
005C 02F0 00241                 decfsz  COUNTER, f      ; decrement counter register
005D 0A56 00242                 goto    TransmitNext    ; transmit next bit
005E 0565 00243                 bsf     PORTA, TX       ; generate Stop bit
005F 0961 00244                 call    DelayFullBit    ; delay for Stop bit
0060 0800 00245                 retlw   0x00            ; return to main routine
                  00246
                  00247 ;*********************************************************************
                  00248 ; * Delay routine 16-bit counter (delay for full bit time)          *
                  00249 ;*********************************************************************
0061 0209 00250 DelayFullBit    movf    AUTOBAUD_HIGH,w ; copy content of Autobaud high register into
0062 0743 00251                 btfss   STATUS, Z       ; is high byte = 0?
0063 0A65 00252                 goto    LoadHighByte    ; no, high byte is not zero
0064 0A6B 00253                 goto    DecLowByteOnly  ; decrement only low byte
                  00254
0065 002E 00255 LoadHighByte    movwf   TEMP2           ; load TEMP2 with content of AUTOBAUD_HIGH
0066 006D 00256                 clrf    TEMP1           ; reset TEMP1 register
0067 02ED 00257 DecLowByte1     decfsz  TEMP1, f        ; decrement low byte
0068 0A70 00258                 goto    DecLowByte11    ; do until result is zero
0069 02EE 00259                 decfsz  TEMP2, f        ; decrement low byte
006A 0A67 00260                 goto    DecLowByte1     ; decrement low byte again
                  00261
006B 0208 00262 DecLowByteOnly  movf    AUTOBAUD_LOW, w ; copy low byte from autobaud register
006C 002D 00263                 movwf   TEMP1           ; into TEMP1
006D 02ED 00264 DecLowByte2     decfsz  TEMP1, f        ; decrement low byte until zero
006E 0A71 00265                 goto    DecLowByte22    ; extra two cycle delay
006F 0800 00266                 retlw   0x00            ; return from subroutine
0070 0A67 00267 DecLowByte11    goto    DecLowByte1     ; additional two cycle delay
0071 0A6D 00268 DecLowByte22    goto    DecLowByte2     ; additional two cycle delay
```

```
                  00269
                  00270
                  00271                 ; *********************************************************************
                  00272                 ; * Delay routine 16-bit counter (delay for half bit time)            *
                  00273                 ; *********************************************************************
0072 020B         00274 DelayHalfBit    movf    AUTOHALF_HIGH,w ; copy content of Autobaud high register into
0073 0743         00275                 btfss   STATUS, Z       ; is high byte = 0?
0074 0A76         00276                 goto    LoadHighByteH   ; no, high byte is not zero
0075 0A7C         00277                 goto    DecLowByteOnlyH ; decrement only low byte
                  00278
0076 002E         00279 LoadHighByteH   movwf   TEMP2           ; load TEMP2 with content of AUTOHALF_HIGH
0077 006D         00280                 clrf    TEMP1           ; reset TEMP1 register
0078 02ED         00281 DecLowByteH1    decfsz  TEMP1, f        ; decrement low byte
0079 0A81         00282                 goto    DecLowByteH11   ; do until result is zero
007A 02EE         00283                 decfsz  TEMP2, f        ; decrement low byte
007B 0A78         00284                 goto    DecLowByteH1    ; decrement low byte again
                  00285
007C 020A         00286 DecLowByteOnlyH movf    AUTOHALF_LOW, w ; copy low byte from autobaud register
007D 002D         00287                 movwf   TEMP1           ; into TEMP1
007E 02ED         00288 DecLowByteH2    decfsz  TEMP1, f        ; decrement low byte until zero
007F 0A82         00289                 goto    DecLowByteH22   ; extra two cycle delay
0080 0800         00290                 retlw   0x00            ; return from subroutine
0081 0A78         00291 DecLowByteH11   goto    DecLowByteH1    ; additional two cycle delay
0082 0A7E         00292 DecLowByteH22   goto    DecLowByteH2    ; additional two cycle delay
                  00293
                  00294
                  00295                 END

Program Memory Words Used:   132
Program Memory Words Free:   380
```

AN715

Measure Tilt Using PIC16F84A & ADXL202

Author: *Rodger Richey*
Microchip Technology

INTRODUCTION

Recent advances in accelerometer sensor technology, especially with silicon micromachined types, have driven the cost of these devices down significantly. As of today, you could obtain an accelerometer for less than $5 per axis. Measurement of acceleration or one of the derivative properties such as vibration, shock, or tilt has become very commonplace in a wide range of products. At first you might think of seismic activity or machinery performance monitoring, but would automotive airbags, sports training products, or computer peripherals ever cross your mind? The technology behind acceleration sensors has advanced to provide a very cost effective and user friendly solution for almost any application.

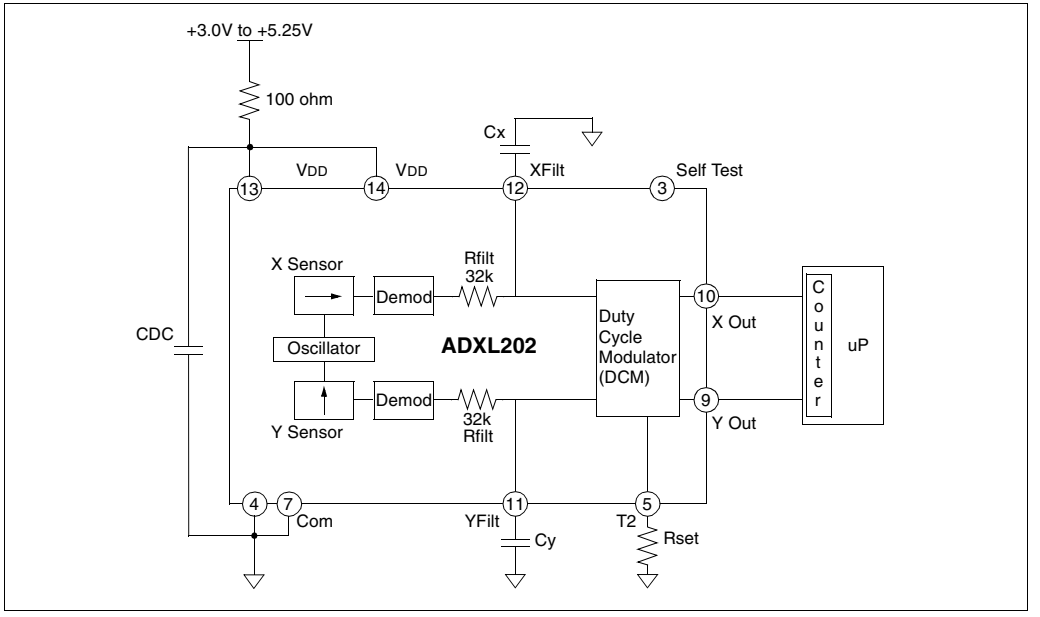
There are many types of sensors that measure acceleration, vibration, shock, or tilt. These sensors include piezo-film, electromechanical servo, piezoelectric, liquid tilt, bulk micromachined piezo resistive and capacitive sensors, as well as surface micromachined capacitive. Each of these sensors has distinct characteristics in the output signal of the sensor, cost to develop, and type of operating environment. Measurement of acceleration can also provide velocity by single integration and position by double integration. Vibration and shock can be used for machine health determination as well as motion and shock detection for car alarms. Static acceleration due to gravity can be used to determine tilt and inclination provided that the sensor is responsive to static acceleration.

This application note will focus on the surface micromachined capacitive ADXL accelerometers from Analog Devices, in particular the ADXL202. The example application will use the ADXL202 accelerometer with the PIC16F84A in a tilt meter. The PIC16F84A is a good match with the ADXL202 because all acceleration measurements are digital only. Secondly, the Data EEPROM can be used to store the calibration constants and restore on reset. The external interface can also be changed easily to accommodate a LCD display (as shown in this application note) or a serial interface to the outside world.

MEMs SENSOR: THEORY OF OPERATION

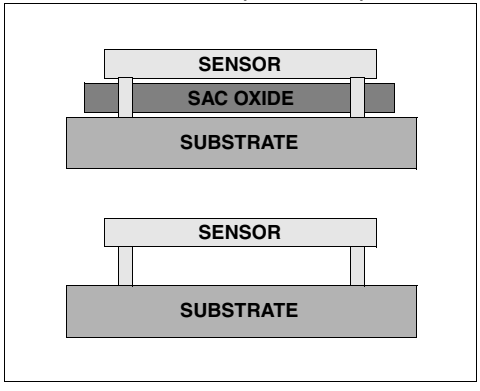
In recent years the silicon micromachined sensor has made tremendous advancements in terms of cost and level of on-chip integration for acceleration and/or vibration measurements. By implementing additional BiMOS circuitry on-chip, these products not only provide sensor but also signal conditioning in a single package that requires a few external components to complete the circuit. Some manufacturers have taken this approach one step further by converting the analog output of the sensor to a digital format such as duty cycle. This method not only lifts the burden of designing fairly complex analog circuitry for the sensor but also reduces cost and board area. Because of these advances, the micromachined accelerometer is finding its way into such products as joysticks and airbags that were previously impossible due to price or size limitations of the sensor. Figure 1 shows the block diagram of the ADXL202.

FIGURE 1: ADXL202 BLOCK DIAGRAM



A surface micromachined device is composed of springs, masses and motion sensing components. These sensors use standard integrated circuit processing techniques in standard wafer fabs, i.e., no additional cost to the user for special processes or fabs. As shown in Figure 2, normal IC processes take place by applying layers of oxide and polysilicon. Then using IC photolithography and selective etching the sensor is created as a 3-dimensional structure suspended above the substrate free to move in all directions. The surrounding area becomes the signal conditioning and output circuitry.

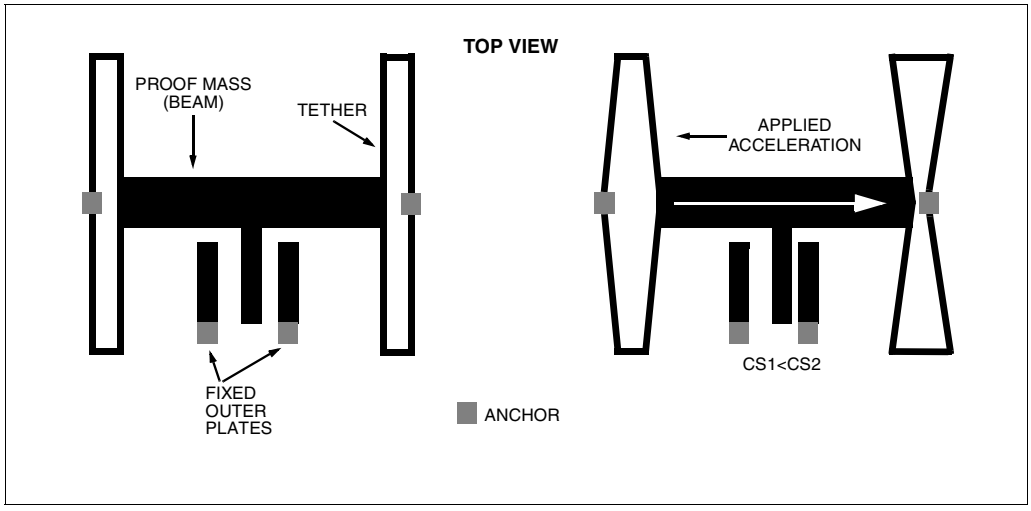
FIGURE 2: SILICON STRUCTURE OF ADXL202 (SIDE VIEW)



The core of the sensor is a surface micromachined polysilicon structure or mass that is suspended on top of the silicon wafer for each axis. The polysilicon "springs" hold the mass and provides resistance to movement due to acceleration forces. Both the mass and the substrate have plates that form a differential capacitor where the fixed plates on the substrate are driven 180° out of phase. Figure 3 shows an exaggerated diagram of the sensor. Any movement of the mass unbalances the differential capacitor resulting in a square wave output with the amplitude proportional to the acceleration. Each axis has a demodulator that rectifies the signal and determines the direction of the acceleration. This output is fed to a duty cycle modulator (DCM) that incorporates external capacitors to set the bandwidth of each axis. The analog signal is filtered and converted to a duty cycle output by the DCM. An external resistor sets the period of the duty cycle output. A 0g acceleration produces a 50% duty cycle output. A low-cost, all digital, microcontroller can be used to measure acceleration by timing both the duty cycle and the period of each axis. Refer to Figure 1 for interaction and connections between the various circuits inside the device as described above.

Some of the advantages with micromachined sensors are that they are low cost and most have on-chip signal conditioning.

FIGURE 3: SENSOR MECHANICAL OPERATION



CONFIGURING THE ADXL202

The end application for our ADXL202 is a simple tilt meter that shows X-axis and Y-axis tilt (or pitch and roll for you aeronautical buffs). The design procedure is somewhat iterative since the bandwidth, period, and microcontroller counter resolution play important roles in the minimum resolution of the measurement. Analog Devices has simplified the design procedure by providing an Excel spreadsheet entitled "The XL202 Interactive Designer" that can be downloaded off their website at www.analog.com and is shown in Appendix A. The specifications for our system are +5VDC operation, +/-1.0 degree tilt resolution, 25 samples per channel per second, and the microcontroller should operate at 4MHz or less.

Through the use of an iterative process, the designer can determine the external component values and the noise and resolution of the acceleration measurement without having to prototype a single circuit.

In Step 1of the spreadsheet shown in Appendix A, the designer will enter the supply voltage which should be between 3.0V and 5.25V. We will enter 5.0V. Analog bandwidth is entered in Step 2, which calculates the values for the external capacitors. The bandwidth directly determines the noise floor and resolution of the accelerometer and therefore may have to be adjusted to provide the desired results based on calculations later in the spreadsheet. Enter 10Hz and the resulting capacitance is 0.50µF. Since 0.50µF is not a standard, we can modify the bandwidth to get a standard value. Using 10.5Hz yields a capacitor of 0.47µF.

In Step 3, the spreadsheet calculates the RMS and peak-to-peak (P-P) noise of the acceleration measurements. The designer must estimate the amount of time that the actual signal will be above the P-P noise using a multiplier. At this step enter 4, which in turn reveals that the peak-to-peak noise will be 0.46 degrees of tilt. Now the designer must evaluate the P-P noise estimation because this noise determines the smallest acceleration resolution that the accelerometer can have. If this noise estimation is not acceptable, then the bandwidth must be lowered to reduce the P-P noise. This example is well within the 1.0 degree specification and so we will continue.

The next few steps set the period of the duty cycle output and the measurement resolution due to the counter on the microcontroller. Both the sample rate per channel and the percentage of time the ADXL202 will be powered are entered in Step 4. The designer also enters the time required to calculate the acceleration for two channels and the spreadsheet then calculates the period of the duty cycle output and the corresponding external resistor. We will use 25 samples per second per channel and the part will be powered up 100% of the time. Analog Devices has already calculated the time to acquire two channels and perform the calculations as 20ms. Of this time, it takes 3ms for calculations based on a previous application note, leaving 17ms for signal acquisition. This relates to 17,000 instruction cycles on the PICmicro® running at 4MHz.

In Step 5, the counter rate of the microcontroller is used to calculate the measurement resolution due to the counter in g's and degrees of tilt. The spreadsheet also determines the size of the counter on the microcontroller to prevent an overflow. Per our specifications, the microcontroller is clocked at 4MHz resulting in a 1MHz timer frequency (Timer0). With this timer rate, the resolution of the digital section of the ADXL202 is 0.06 degrees of tilt. The counter required to acquire the digital output must be 15-bits. We can easily implement a 15-bit counter using the Timer0 as the low byte of the count and for each Timer0 overflow increment an upper byte counter. The designer must again determine if this

resolution is acceptable. To increase the resolution, either increase the counter rate (Step 5) or decrease the number of samples per second (Step 4).

Step 6 checks for aliasing errors due to the sample rate. Nyquist requirements specify that the sample rate needs to be faster than the bandwidth by a factor of 2. Analog Devices recommends that at least a factor of 10 is used to minimize dynamic errors from the PWM sampling technique. For our case the, the ratio is 11.2 which according to Nyquist and Analog Devices is more than sufficient. If the spreadsheet calculates that the ratio is low, the designer must increase the sample rate in Step 4 or decrease the bandwidth in Step 2.

The results are in! The spreadsheet calculates the minimum resolution of the acceleration measurement due to RMS P-P noise and resolution of the counter in Step 7. It also provides a minimum resolution of a tilt measurement. Our calculated minimum resolution is 0.5 degrees of tilt which is acceptable according to the specification. If this resolution was not acceptable, then the bandwidth (Step 2), acquisition rate (Step 4), or the counter rate (Step 5) would have to be adjusted to reduce noise.

The spreadsheet also offers the designer the ability to explore how oversampling the PWM signal affects noise at the expense of sacrificing bandwidth in Step 8. Finally, Step 9 provides the estimated drift of the 0 g point due to temperature effects.

TILT METER APPLICATION

In the tilt meter application, the value of tilt in the X-axis and Y-axis is displayed on a 2-line by 8-character dot matrix LCD display. The only other function is a push button switch to perform a simple calibration cycle. The PIC16F84A makes an ideal companion to the ADXL202 because calibration parameters for the sensor can be stored in on-chip Data EEPROM memory for retrieval and usage in later calculations. Using the ADXL202 in conjunction with a PICmicro® not only reduces the time to market for the product but also overall system cost and power consumption.

Figure 4 shows the schematic for the simple tilt meter. For convenience, a 9V battery is used with a LM78L05 +5V regulator to provide power to the circuit. The ADXL202 is configured as shown in the ADXL202 Interactive Designer spreadsheet with 0.47µF capacitors on the XFILT and YFILT pins. A resistor of 1.0625MΩ is called out by the spreadsheet to connect to the RSET pin. Since the duty cycle generator's current source that determines the PWM frequency is only accurate to approximately 10%, a 1.0 MΩ, 5% resistor can be used. The 6% error between the two resistors will get corrected by the measurement of T2. The duty cycle output pins from the ADXL202 XOUT and YOUT are connected to RA0 and RA1 respectively.

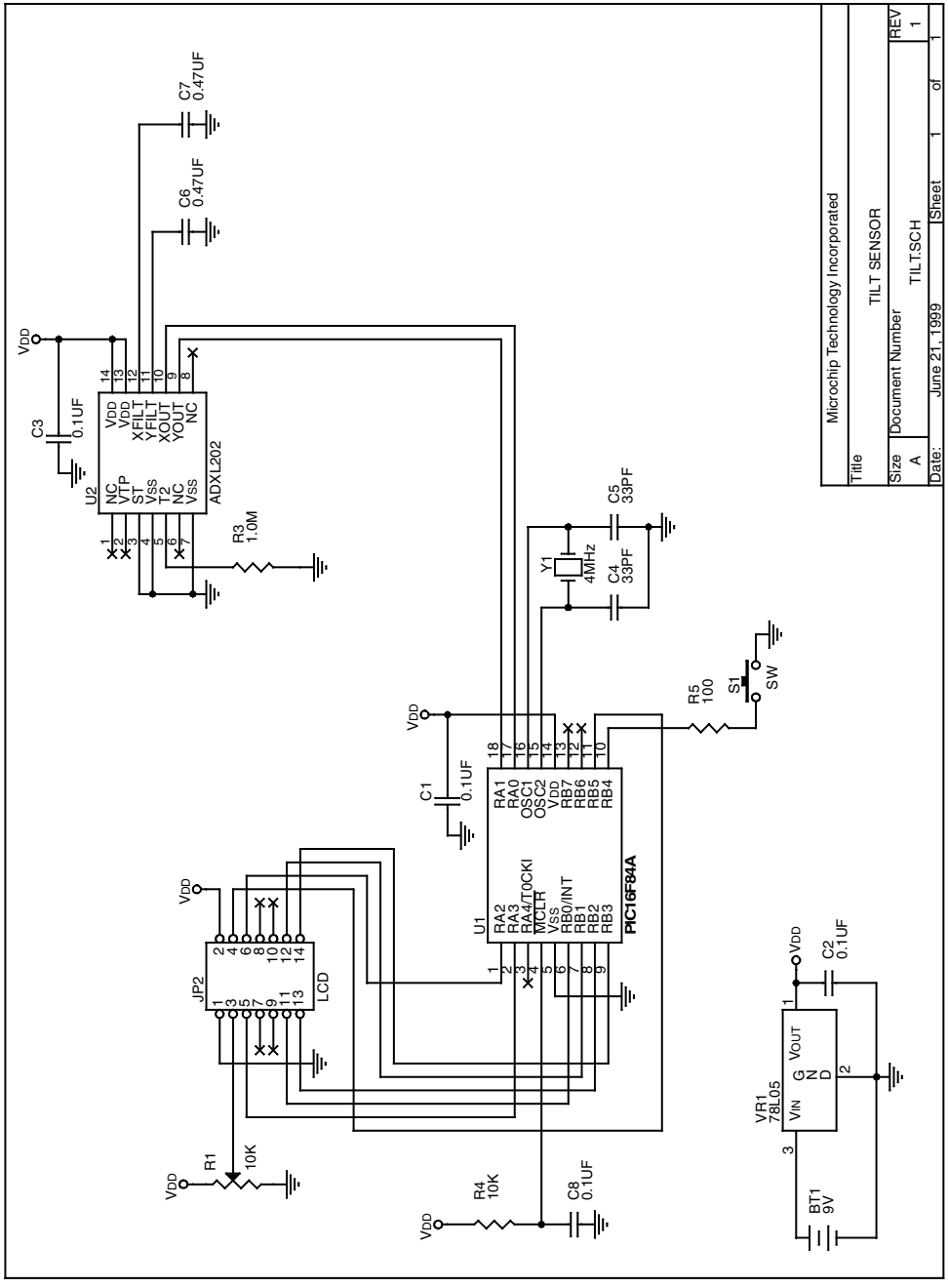


FIGURE 4: TILT METER SCHEMATIC

The microcontroller circuit is also very simple. The 4MHz crystal uses two 33pF capacitors to complete the oscillator circuit. The push button switch is connected to RB4. This pin has internal pull-up resistors reducing the need for any external circuitry. The LCD display is driven using the 4-bit MPU mode which only requires 3 I/O pins for control and 4 I/O pins for data. Refer to the specifications for the Hitachi HD44780 LCD controller or the application note, AN587, "Interfacing PICmicros® to an LCD Module", for more interface information. The controls lines RS, R/W, and E connect to RB5, RA3, and RA2. The data lines are connected to RB<0:3>. There is also a 10KΩ potentiometer connected to pin 3 of the LCD display to control the contrast.

Acceleration is a vector quantity with both a direction and magnitude. The acceleration vector can be broken up into two vectors on the ADXL202, the X-axis and Y-axis. The ADXL202 is responsive to both static acceleration due to gravity as well as acceleration due to motion. The main problem with using this type of accelerometer to measure degrees tilt is not that it is sensitive to motion but that it can't distinguish between gravity and motion. The user must implement some type of time weighted filter to remove the effects of motion from the measurement (not implemented in this design). When the tilt angle is varied along the sensitive X- and Y-axis, the acceleration vector changes and the ADXL202 responds by changing the duty cycle outputs. The angle of tilt is defined by the following equation:

$\theta = arcsin[(V(out)-V(zero\ g)) / (1g \times Scale\ factor(V/g))]$

This is a difficult calculation on an 8-bit microcontroller, therefore the calculation will be simplified. In spite of this, we still yield very good results (shown later in the firmware section).

The firmware is centered around the duty cycle measurement. The technical note from Analog Devices titled "Using the ADXL202 Duty Cycle Output" shows a very efficient method of measuring the period and duty cycle of the PWM waveforms. Figure 5 shows the waveforms from XOUT and YOUT. The most obvious method of measuring these waveforms is to measure the time from rising edge to falling edge to next rising edge for each of the waveforms. While very simple, this method takes 3 complete cycles to complete the process. If you take a closer look at Figure 5, you will see that the high time of XOUT and YOUT are centered about each other. We can use this to our advantage.

Figure 6 shows the waveform and measurement points for the improved measurement scheme. At Ta the counter is started. The program then records the times at Tb, Tc, and Td. By looking at the points of Figure 6, we can say that:

T1x = Tb - Ta = Tb (counter was 0 at Ta)

T1y = Td - Tc

T2x = T2y = T2 = Te - Ta = Tg - Tf

Since we have already established that the center of T1 is aligned with the center of T2, the equation for T2 reduces to:

T2 = Td - T1y/2 - T1x/2

T2 = Td - (Td - Tc)/2 - Tb/2

This technique not only reduces the measurement time to two cycles, it also only calculates T2 once.

FIGURE 5: ADXL202 DUTY CYCLE OUTPUT

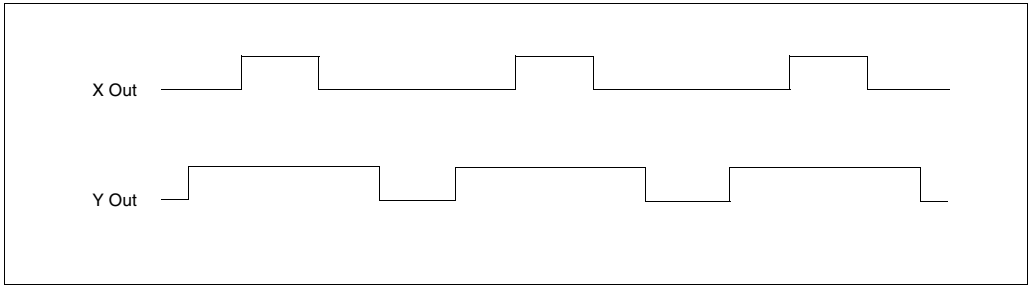
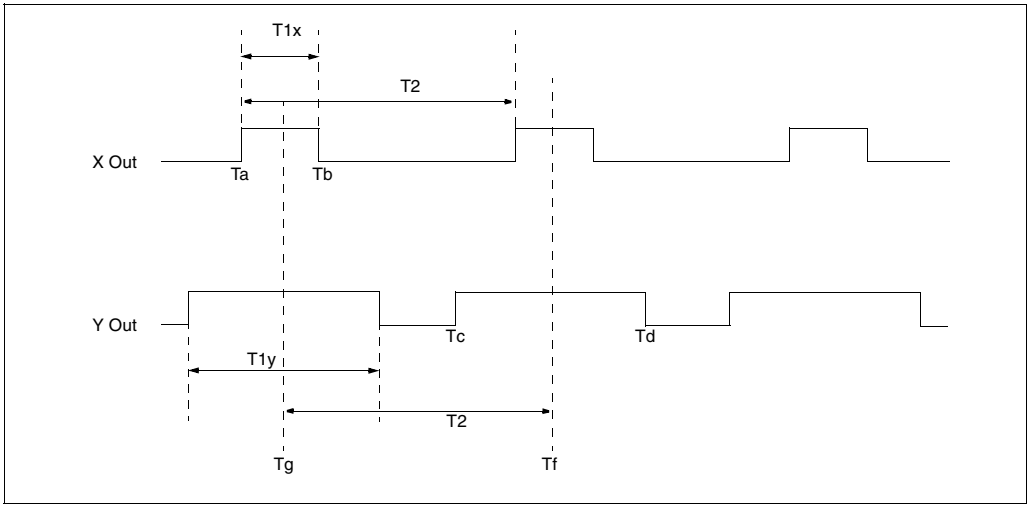


FIGURE 6: ADXL202 DUTY CYCLE MEASUREMENT



Now that our system is reading the duty cycle outputs of the ADXL202 and displaying the results on the LCD display, we need to consider how the system is calibrated. The first calibration step is the initial calibration of the tilt sensor with respect to gravity. The simplest method is to position the system such that the X-axis and Y-axis are both level. When instructed to calibrate, the PIC16F84A will calculate the duty cycle output T1 for both axis and the period T2. Several readings may be taken and averaged to improve the accuracy of the measurements. These values are now stored in both RAM as well as EEPROM as calibration constants. A scale factor is also used in the calibration process to create an n-bit result. These constants are defined as:

- T2cal, the value of T2 during the calibration phase. T2 must be stored because it can vary over temperature and has jitter from one measurement to another.
- ZXcal, the value of T1x during the calibration phase.
- ZYcal, the value of T1y during the calibration phase.
- K, the scale factor and is equal to *[4 * (T2cal * bit_scale_factor) / T2cal]*

K needs to be calculated only once. Since each axis will use this factor it is hard coded in the firmware. The bit_scale_factor is used to determine the size of the result. Since we are looking for a result of +/- 90 (1 degree of tilt per count), the bit scale factor would be 180. Therefore, K is assigned a value of 720. This is the simplification that was mentioned earlier.

Once we have calculated the calibration constants we can apply them to the duty cycle measurements to get degree of tilt. The following formulas give the degree of tilt for each axis:

*ZXactual = (ZXcal * T2actual) / T2cal (1)*

*ZYactual = (ZYcal * T2actual) / T2cal (2)*

T2actual is the current measurement of T2. This formula adjusts the 0g value for changes in T2 due to temperature or jitter.

*XAcceleration = [K * (T1x - ZXactual)] / T2actual (3)*

*YAcceleration = [K * (T1y - ZYactual)] / T2actual (4)*

The values of T1x and T1y are the current measurements of T1 for each axis. The results in XAcceleration and YAcceleration are the degrees to tilt in the X-axis and Y-axis directions properly scaled for 1 degree per count. This method of calibration is very simple yet will suffer from small errors due to variance in duty cycle % per g (which was assumed to be 12.5%) from one part to the next.

The order of the math operations is deliberate to preserve the accuracy of the result. All math operations are done in fixed point math. Several variables are used in the math operations. The following table shows the two inputs to each routine and the location of the result of the routine.

TABLE 1: MATH OPERATIONS VARIABLE USAGE

Operation	Operand #1	Operand #2	Result
16 x 16 Addition	ACCHI, ACCLO	ARGH, ARGL	ACCHI, ACCLO
16 x 16 Subtraction	ACCHI, ACCLO	ARGH, ARGL	ACCHI, ACCLO
16 x 16 Multiply	ACCHI, ACCLO	ARGH, ARGL	PRODW3, PRODW2, PRODW1, PRODW0
32 x 16 Divide	PRODW3, PRODW2, PRODW1, PRODW0	DIV1, DIV0	ANS1, ANS0

For calculating Zactuals in formula (1) and (2), the 16 x 16 multiply of Zcal * T2actual takes place first followed by the division of the result by T2cal. When calculating tilt (really is scaled acceleration) in the formulas (3) and (4), the subtraction of Zactual from T1 takes place first, followed by multiplication of the result by K, and finally the division of the result by T2actual.

Finally, the last two pieces of the code are for the LCD display and the Data EEPROM access. The LCD code is a derivative of that found in the application note, AN587, "Interfacing PICmicro® Microcontrollers to an LCD Module". Most of the changes were related to the different I/O pins used for data and control. The Data EEPROM routines use code directly from the PIC16F84A data sheet DS35007 for reads and writes. The WriteCal routine takes the calibration constants and writes them to the Data EEPROM. This routine is only called when a calibration cycle is performed. The RestoreCal routine is called when the PIC16F84A is reset. The calibration constants are grouped sequentially in memory so that these routines can use indirect addressing and shorten the length of code.

CONCLUSION

As in all applications, the type of acceleration sensor depends on the system requirements as well as the property being measured. Some accelerometers are better suited towards measuring vibration and shock such as the piezo-film and piezoelectric. Others are used for tilt measurements such as liquid tilt and micromachined types. The type of sensor selected then dictates the signal conditioning circuitry requirements. Some accelerometers have AC response, some DC. Some sensors have analog outputs, others digital. In other words, one accelerometer does not fit into all applications. This application note has shown that a designer can quickly and easily complete an accelerometer based design using the ADXL sensors from Analog Devices. The use of a microcontroller further simplifies the design by giving the designer a more integrated, lower cost solution for the data measurement application.

REFERENCES

Data Sheets

- *ADXL202 Data Sheet,* Analog Devices Inc., Rev. 0
- *ADXL210 Data Sheet,* Analog Devices Inc., Rev. Pr.A
- *PIC16F84A Data Sheet,* Microchip Technology Inc., DS35007A

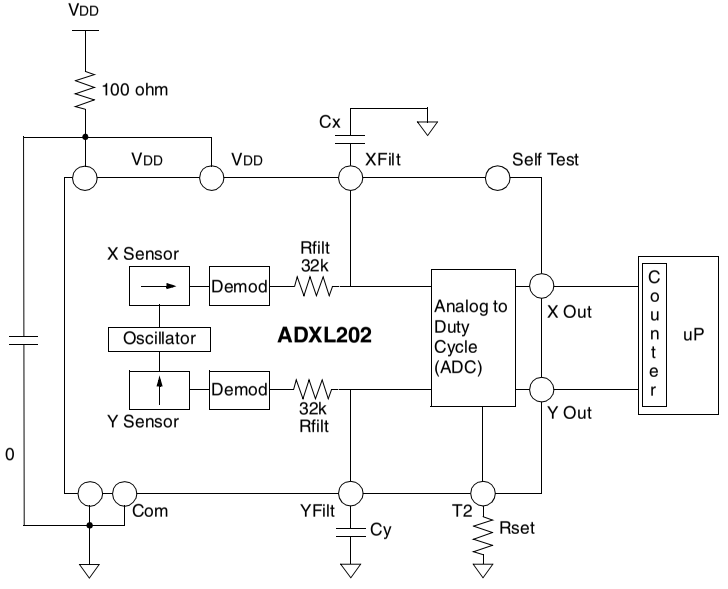
Technical Notes From Analog Devices

- *Using the ADXL202 Duty Cycle Output*
- *Accelerometer Design and Applications*
- *A Compact Algorithm Using The ADXL202 Duty Cycle Output*
- *The Interactive Designer ADXL202*

APPENDIX A: ADXL202 INTERACTIVE DESIGNER

The XL202 Interactive Designer

Enter values below. When your design is complete the values for your design will print out on this page.



Parameters	
Supply voltage	5.0 V
Analog Bandwidth	10.5 Hz
Acquisition Rate	25 readings per second
Resolution (g's)	0.008 g
Resolution (deg of tilt)	0.47 deg of tilt
Microcontroller counter rate	1 MHz
T2	8.5 mS
Power cycling %	100% on time
Tmax	35 deg C
Tmin	15 deg C
Zero g drift Tmax	0.02 g
Zero g drift Tmin	0.02 g

Component Values	
Supply Decoupling	0.1 uF
Xcap, Ycap	0.47 uF
Rset	1062.5 kohm

You will be asked to enter variety of design parameters important to your applications. This will include such issues as how fast is the signal you need to measure, what is the required update or acquisition rate, what is the counter speed on your microcontroller. After entering target values (inputs) the spreadsheet will calculate outputs such as the resolution of the accelerometer. You can then iterate the input values and trade off parameters as necessary to meet your design goals. Only enter values that are in **bold**.

1. Enter your nominal supply voltage

The XL202 will operate from 3.0V to 5.25V. Enter your nominal supply voltage here.

Vdd **5.0** **V**

2. What is the fastest signal you want to be able to observe?

In this step you will determine the bandwidth for the analog stage of the accelerometer. The bandwidth generally determines the noise floor and thus the resolution of the accelerometer. In a later section you will also calculate digital noise sources from the PWM stage; the combination of these two noise sources determines the total noise floor. You will be measuring a real world acceleration, such as human or vehicle motion. What part of the signal content is important? If the signals are transient, such as shock or impulse, you may want to set a higher bandwidth. Human motion can often be measured at 10Hz or less. Don't forget to consider filter delays that could result in a lag between a stimulus and a response by the accelerometer, (dominated by the filter). Component values for the Xfilt and Yfilt capacitor are calculated below. You will probably want to iterate to a standard capacitor value.

Enter desired Bandwidth **10.5** Hz Value for Cx, Cy 0.47 uF Component Value!

3. Estimate P-P noise

The peak to peak noise of the accelerometer is the best indicator of resolution of the accelerometer. Noise is a statistical process, and is best described by an RMS measurement, (available on the datasheet). P-P noise is then estimated using a statistical estimation. You need to select a RMS to P-P estimation. The table below tells you how various RMS to P-P noise multipliers, predict the amount of time the actual signal will EXCEED the estimated P-P noise. The lower the multiplier, the more likely it is that a noise event will exceed the P-P limit.

Enter RMS to P-P multiplier **4** X of RMS

RMS Multiplier	% of time a signal will exceed the P-P estimate
2X	32.00%
4X	4.60%
6X	0.27%
8X	0.01%

Calculated noise at the analog output Xfilt and Yfilt

Noise(rms) at Xfilt, Yfilt	0.002 g (max RMS)	
Noise(P-P) at Xfilt, Yfilt	0.008 g (max P-P) @4X RMS	
Noise(P-P) at Xfilt, Yfilt	0.47 Deg of tilt (max P-P)	At 17mg/deg of tilt

Note: Noise level is inversely proportional to supply voltage
Note Decrease Noise (increase resolution) by decreasing BW.

3A. Iterate

Look at the P-P noise estimate; this is the noise limited resolution, (the smallest signal you can resolve). Is this acceptable for your application? If not you should consider adjusting the bandwidth down to reduce P-P noise and improve resolution.

4. How fast would you like to acquire the signals?

In this section we will begin the design of the digital output, and the microcontroller interface. You will input an acquisition rate, i.e. how many times per second you want a new reading from the accelerometer. You are also asked how long the part should be powered each second. Note that if you only want a few samples per second, but intend to keep the part powered all of the time, then you will need to set a faster acquisition rate in order to get reasonable values for the PWM output. The program requests that you input the time required to do the multiplies and divides to calculate the acceleration. 3.0ms is the time required for a Microchip 16C63 running at 4 Mhz. This section generates a component value for the Rset resistor.

Enter desired acquistion rate **25** Each Channel per second

% of time part will be powered per second **100%**

Calculate Acquisition Time

Maximum time available to acquire 2 channels	20.0	ms
Time required to calculate two channels	**3.0**	ms
Time left for signal acquisition	17.0	ms (two channels)

This implies a requirement for the value of the PWM period T2

Thus, T2 =	8.5 mS or	118 Hz	This is the Sample Rate
Value for Rset	1062.5 kohm		Component Value!

5. Enter the counter rate of your Microcontroller and calculate the resolution of the digital output.

In Section 2, we calculated the resolution of the analog section. In this section we will calculate the resolution of the digital output; a fuction of the PWM rate T2 (calculated in section 4), and the counting rate of your microcontroller. Please note that the counting rate is different, and usually slower than the microcontroller clock rate. The output of this calculation is a measure of the quantization error of the counter. In some cases it may limit the ultimate resolution; we will explore this in

Counter Rate **1 Mhz**

Note: you will need a counter of size 17000 counts or 15 bits
To avoid overflowing the counter

Resolution	1062.5 Counts per g	
Resolution	0.001 g	Quantization bit size
Resolution	0.06 Deg of Tilt	Based on 17mg/deg of tilt

Note: Increase resolution by increasing counter rate or decreasing samples per second

6. Check for aliasing and other errors in sampling:

In all cases the sample rate (1/T2) needs to be faster than the bandwidth of the analog section by a factor of at least 2 in order to meet the requirements of Nyquist. Nyquist notwithstanding, a ratio of at least 10 is recommend to minimize dynamic errors that are endemic to PWM sampling techniques. If your ratio is low, you can improve it by either increasing the sample rate (by increasing the acquistion rate in section 4) or decreasing the analog bandwidth (in section 2).

Ratio of sample rate (1/T2) to analog BW: 11.2 Good!

7. Estimate of total resolution (iterate to meet design objective)

We are now in a position to bring together the various calculation above to determine the resolution of the complete analog and digital design. The ultimate resolution is determined by both the noise at the analog output (Xcap and Ycap) and the quantization bit size of the PWM + counter system. At this point check the total system resolution to see if it meets your requirements. If it does not, then revisit bandwidth at Xfilt and Yfilt, acquistion rate or counting rate to reduce noise. You may also want to consider digital filtering, (oversampling) to reduce noise at the expense of sampling rate as discussed in the next section.

Noise due to analog section	0.008 g (max P-P) @4X RMS	This is the noise contribution at the analog output Xcap, Ycap
Resolution of digital output- counter	0.001 g	This is the quantization noise of the digital output
Estimated Total Noise (resolution):	0.008 g P-P	This is the total P-P noise, which is the root sum square of the analog and digital noise.
Estimated Total Noise (resolution):	0.5 deg of tilt @ 17mg/deg	
Noise (Resolution) is limited by:	Bandwidth at Xfilt, Yfilt; reduce bandwidth if lower noise desired (section 2)	

8. Option: Reduce noise by oversampling (at expense of bandwidth)

Another design option is to use digital filtering (averaging) in order to reduce noise, at the expense of bandwidth. By averaging several samples you are in effect filtering the signal. Implementing averages of 2,4 8, 16 samples are simple right shifts in microcontroller code (very efficient). For oversampling to work, samples need to be taken at a rate no faster than 10 times the analog bandwidth. Note: make sure oversampling is set to 1 sample if you don't want to use oversampling!

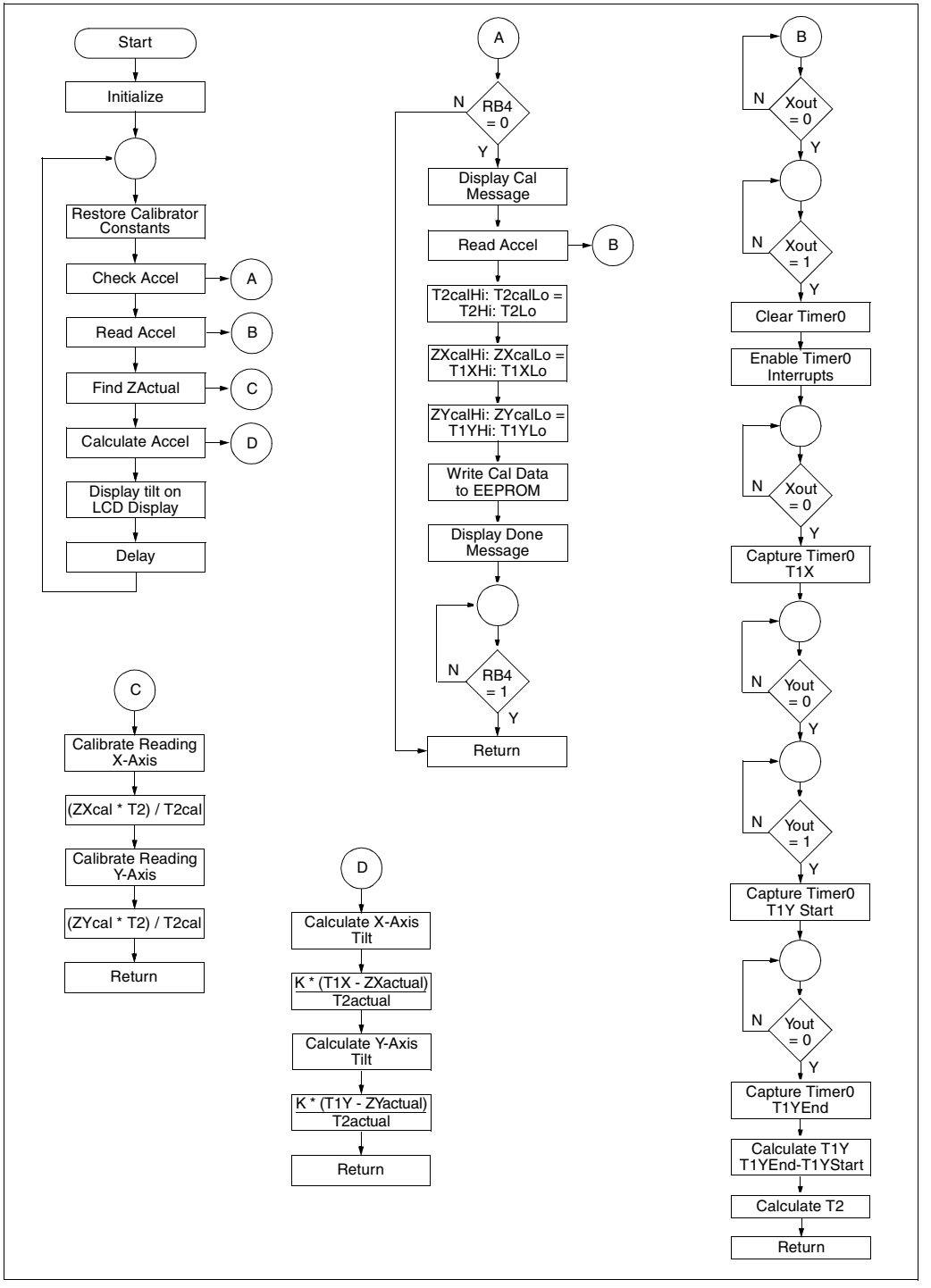
Estimated Noise (resolution) with average of:	1	Samples	Noise before oversampling	0.008 g P-P	
Note: Samples should be taken about:	95.2381	mS apart	Noise after oversampling	0.008 g P-P	0% Reduction
			Bandwidth before oversampling	10.5 Hz	
			Bandwidth after oversampling	10.5 Hz	

9. Estimated Drift of Zero g point

You can estimate the zero g temperature shift by entering your expected temperature range below and an estimate of the drift in mg/C (from the data sheet). Note that zero g drift can be positive or negative, but in general is very linear. X axis and Yaxis drift are uncorrelated.

				Drift from 25C Value		
				Drift in mG	Drift in deg of tilt	
Drift in mG per degree C	**0.002**	g/C				
Max Temp	**35**	C	at Tmax	0.02 mg	1.2 deg of tilt	at 17mg/deg C
Min Temp	**15**	C	at Tmin	0.02 mg	1.2 deg of tilt	at 17mg/deg C

APPENDIX B: TILT SENSOR FIRMWARE FLOWCHART



Please check Microchip's Worldwide website at www.microchip.com for the latest version of the source code.

APPENDIX C: TILT MOTOR SOURCE CODE LISTING

```
           00001         list    p=16f84a
           00002         include <p16f84a.inc>
           00001         LIST
           00002 ; P16F84A.INC Standard Header File,Version 2.00 Microchip Technology
           00134         LIST
           00003
2007 3FF1  00004         __config    _CP_OFF&_WDT_OFF&_XT_OSC&_PWRTE_ON
           00005 ;Assembled using MPASM V2.30
           00006 ;PORTA defines
           00007 #define XOUT    0
           00008 #define YOUT    1
           00009 #define E       2
           00010 #define RW      3
           00011
           00012 ;PORTB defines
           00013 #define CAL     4
           00014 #define RS      5
           00015
           00016 ;=============================================================================
           00017 ;       RAM EQUATES
           00018 ;=============================================================================
           00019         cblock  0x0c
 0000000C  00020                 T1XHi
 0000000D  00021                 T1XLo
 0000000E  00022                 ArgL
 0000000F  00023                 ArgH
 00000010  00024                 AccHi
 00000011  00025                 AccLo
 00000012  00026                 DivCnt
 00000013  00027                 PRODW3
 00000014  00028                 PRODW2
 00000015  00029                 PRODW1
 00000016  00030                 PRODW0
 00000017  00031                 DIV0
 00000018  00032                 DIV1
 00000019  00033                 ANS0
 0000001A  00034                 ANS1
 0000001B  00035                 T2Hi
 0000001C  00036                 T2Lo
 0000001D  00037                 T1YStartLo
 0000001E  00038                 T1YStartHi
 0000001F  00039                 T1YEndLo
 00000020  00040                 T1YEndHi
 00000021  00041                 T1YHi
 00000022  00042                 T1YLo
 00000023  00043                 ZXcalHi
 00000024  00044                 ZXcalLo
 00000025  00045                 ZYcalHi
 00000026  00046                 ZYcalLo
 00000027  00047                 T2calHi
 00000028  00048                 T2calLo
 00000029  00049                 ZXActualHi
 0000002A  00050                 ZXActualLo
 0000002B  00051                 ZYActualHi
 0000002C  00052                 ZYActualLo
```

```
 0000002D  00053                XAccel
 0000002E  00054                YAccel
 0000002F  00055                Temp0
 00000030  00056                Temp1
 00000031  00057                Temp2
 00000032  00058                Temp3
 00000033  00059                Timer0H
 00000034  00060                EADR
 00000035  00061                EDATA
           00062         endc
           00063
 0000000E  00064 Count1  equ    ArgL
 0000000F  00065 Count2  equ    ArgH
 0000002F  00066 Temp    equ    Temp0
 00000019  00067 CMD     equ    ANS0
 00000019  00068 LDATA   equ    CMD
 00000017  00069 Digit0  equ    DIV0
 00000018  00070 Digit1  equ    DIV1
           00071
 00000002  00072 KHi     equ    0x02
 000000D0  00073 KLo     equ    0xd0
           00074
0000       00075         org    0x0000
0000 2808  00076         goto   Start           ;Go to start of program
           00077
0004       00078         org    0x0004
0004 0AB3  00079         incf   Timer0H,F
0005 110B  00080         bcf    INTCON,T0IF
0006 118B  00081         bcf    INTCON,RBIE
0007 0009  00082         retfie
           00083
0008       00084 Start
0008 1283  00085         bcf    STATUS,RP0
0009 0185  00086         clrf   PORTA
000A 0186  00087         clrf   PORTB
000B 1683  00088         bsf    STATUS,RP0      ;Bank1
000C 3003  00089         movlw  B'00000011'     ;Set up the I/O ports
000D 0085  00090         movwf  TRISA
000E 3010  00091         movlw  B'00010000'
000F 0086  00092         movwf  TRISB
0010 300F  00093         movlw  B'00001111'
0011 0081  00094         movwf  OPTION_REG
0012 1283  00095         bcf    STATUS,RP0      ;Bank0
0013 21E1  00096         call   OpenXLCD        ;Initialize LCD
0014 22C6  00097         call   RestoreCal      ;Restore calibration constants
0015 178B  00098         bsf    INTCON,GIE
           00099
0016       00100 Main
0016 20F2  00101         call   CheckCal        ;Check if need to calibrate
0017 2020  00102         call   ReadAccel       ;Read the acceleration
0018 2060  00103         call   FindZActual     ;Calibrate readings
0019 2085  00104         call   CalculateAccel  ;Calculate tilt (acceleration)
001A 2194  00105         call   DisplayAccel    ;Display results
001B 30FF  00106         movlw  0xff            ;Delay for a while
001C 22A2  00107         call   Delay_Ms_4MHz
001D 30FF  00108         movlw  0xff
001E 22A2  00109         call   Delay_Ms_4MHz
001F 2816  00110         goto   Main            ;Do it again
           00111 ;*****************************************************************
           00112
```

```
                00113 ;*******************************************************************
                00114 ;*********** Acceleration Measurement/Calculation Routines **********
                00115 ;*******************************************************************
                00116 ;*******************************************************************
                00117 ;ReadAccel
                00118 ;       This subroutine acquires and calculates T1X, T1Y, and T2.
                00119 ;               T1X is in registers T1XHi,T1XLo
                00120 ;               T1Y is in registers T1YHi,T1YLo
                00121 ;               T2 is in registers T2Hi,T2Lo
                00122 ;*******************************************************************
0020            00123 ReadAccel
0020            00124 EDGE1
0020 1805       00125         btfsc   PORTA,XOUT      ;Wait for low on XOUT
0021 2820       00126         goto    EDGE1
0022            00127 EDGE2
0022 1C05       00128         btfss   PORTA,XOUT      ;Wait for high on XOUT
0023 2822       00129         goto    EDGE2
0024 0181       00130         clrf    TMR0            ;Clear Timer
0025 01B3       00131         clrf    Timer0H
0026 110B       00132         bcf     INTCON,T0IF     ;Enable Timer0 overflow interrupt
0027 168B       00133         bsf     INTCON,T0IE
0028            00134 EDGE3
0028 1805       00135         btfsc   PORTA,XOUT      ;Look for falling edge on XOUT
0029 2828       00136         goto    EDGE3
002A 0801       00137         movf    TMR0,W          ;Save Timer0H:TMR0 as T1X
002B 008D       00138         movwf   T1XLo
002C 0833       00139         movf    Timer0H,W
002D 008C       00140         movwf   T1XHi
002E            00141 EDGE4
002E 1885       00142         btfsc   PORTA,YOUT      ;Look a low level on YOUT
002F 282E       00143         goto    EDGE4
0030            00144 EDGE5
0030 1C85       00145         btfss   PORTA,YOUT      ;Look for rising edge on YOUT
0031 2830       00146         goto    EDGE5
0032 0801       00147         movf    TMR0,W          ;Save Timer0H:TMR0 for start
0033 009D       00148         movwf   T1YStartLo      ;of T1Y
0034 0833       00149         movf    Timer0H,W
0035 009E       00150         movwf   T1YStartHi
0036            00151 EDGE6
0036 1885       00152         btfsc   PORTA,YOUT      ;Look for falling edge on YOUT
0037 2836       00153         goto    EDGE6
0038 0801       00154         movf    TMR0,W          ;Save Timer0H:TMR0 as the end
0039 009F       00155         movwf   T1YEndLo        ;of T1Y
003A 0833       00156         movf    Timer0H,W
003B 00A0       00157         movwf   T1YEndHi
003C 128B       00158         bcf     INTCON,T0IE
                00159
003D 0820       00160         movf    T1YEndHi,W      ;Calculate T1Y
003E 0090       00161         movwf   AccHi
003F 081F       00162         movf    T1YEndLo,W
0040 0091       00163         movwf   AccLo
0041 081E       00164         movf    T1YStartHi,W
0042 008F       00165         movwf   ArgH
0043 081D       00166         movf    T1YStartLo,W
0044 008E       00167         movwf   ArgL
0045 210E       00168         call    Sub16x16
0046 0810       00169         movf    AccHi,W
0047 00A1       00170         movwf   T1YHi
0048 0811       00171         movf    AccLo,W
0049 00A2       00172         movwf   T1YLo
```

```
004A 0820  00173         movf    T1YEndHi,W      ;CALCULATE T2
004B 0090  00174         movwf   AccHi           ;2*(T2Hi,T2Lo) = (T1YEndHi:T1YEndLo)+
004C 081F  00175         movf    T1YEndLo,W      ;(T1YStartHi:T1YStartLo)-(T1XHi:T1XLo)
004D 0091  00176         movwf   AccLo
004E 081E  00177         movf    T1YStartHi,W
004F 008F  00178         movwf   ArgH
0050 081D  00179         movf    T1YStartLo,W
0051 008E  00180         movwf   ArgL
0052 2107  00181         call    Add16x16        ;ACCHI,ACCLO=(T1YEndHi:T1YEendLo)+
0053 080C  00182         movf    T1XHi,W         ;       (T1YStartHi:T1YStartLo)
0054 008F  00183         movwf   ArgH
0055 080D  00184         movf    T1XLo,W
0056 008E  00185         movwf   ArgL
0057 210E  00186         call    Sub16x16        ;ACCHI,ACCLO = 2*T2
0058 1003  00187         bcf     STATUS,C
0059 0C90  00188         rrf     AccHi,F
005A 0C91  00189         rrf     AccLo,F
005B 0810  00190         movf    AccHi,W
005C 009B  00191         movwf   T2Hi
005D 0811  00192         movf    AccLo,W
005E 009C  00193         movwf   T2Lo
005F 0008  00194         return
           00195
           00196
           00197 ;*****************************************************************
0060       00198 FindZActual
           00199 ;       This subroutine finds the value of ZActual for the X and Y
           00200 ;       axis.
           00201 ;       Output is ZXActualHi & ZXActualLo for the X-axis and
           00202 ;       ZYActualHi & ZXActualLo for the Y-axis.
           00203 ;*****************************************************************
0060       00204 FindZActual
0060 0823  00205         movf    ZXcalHi,W       ;First the X-axis
0061 0090  00206         movwf   AccHi
0062 0824  00207         movf    ZXcalLo,W
0063 0091  00208         movwf   AccLo
0064 081B  00209         movf    T2Hi,W
0065 008F  00210         movwf   ArgH
0066 081C  00211         movf    T2Lo,W
0067 008E  00212         movwf   ArgL            ;PRODW3,PRODW2,PRODW1,PRODW0 =
0068 211A  00213         call    Mul16x16        ;(ZXCAL_HI,ZXCAL_LO)*(T2HI,T2LO)
0069 0827  00214         movf    T2calHi,W
006A 0098  00215         movwf   DIV1
006B 0828  00216         movf    T2calLo,W
006C 0097  00217         movwf   DIV0
006D 2165  00218         call    Div32x16        ;ANS1,ANS0 = (ZXcal * T2) / T2cal
006E 081A  00219         movf    ANS1,W
006F 00A9  00220         movwf   ZXActualHi
0070 0819  00221         movf    ANS0,W
0071 00AA  00222         movwf   ZXActualLo
0072 0825  00223         movf    ZYcalHi,W       ;Same thing for the Y-axis
0073 0090  00224         movwf   AccHi
0074 0826  00225         movf    ZYcalLo,W
0075 0091  00226         movwf   AccLo
0076 081B  00227         movf    T2Hi,W
0077 008F  00228         movwf   ArgH
0078 081C  00229         movf    T2Lo,W
0079 008E  00230         movwf   ArgL            ;PRODW3,PRODW2,PRODW1,PRODW0 =
007A 211A  00231         call    Mul16x16        ;(ZYCAL_HI,ZYCAL_LO)*(T2HI,T2LO)
007B 0827  00232         movf    T2calHi,W
```

```
007C 0098  00233         movwf   DIV1
007D 0828  00234         movf    T2calLo,W
007E 0097  00235         movwf   DIV0
007F 2165  00236         call    Div32x16        ;ANS1,ANS0 = (ZYcal * T2) / T2cal
0080 081A  00237         movf    ANS1,W
0081 00AB  00238         movwf   ZYActualHi
0082 0819  00239         movf    ANS0,W
0083 00AC  00240         movwf   ZYActualLo
0084 0008  00241         return
           00242
           00243
           00244 ;*****************************************************************
           00245 ;CalculateAccel
           00246 ;       This subroutine performs the acceleration calculation for
           00247 ;       each axis.  The formula is:
           00248 ;               ACCEL = [K * (T1-Zactual) / T2actual]
           00249 ;       Output is XAccel and YAccel
           00250 ;*****************************************************************
0085       00251 CalculateAccel
0085 0829  00252         movf    ZXActualHi,W    ;Check if acceleration is positive
0086 020C  00253         subwf   T1XHi,W         ;or negative by comparing
0087 1C03  00254         btfss   STATUS,C        ;T1X and ZXactual
0088 28A5  00255         goto    CA1             ;Jump if T1X < ZXactual
0089 1D03  00256         btfss   STATUS,Z        ;Test if T1XHI=ZX_ACTUAL_HI
008A 288F  00257         goto    CA2             ;Jump if T1X > ZXactual
008B 082A  00258         movf    ZXActualLo,W
008C 020D  00259         subwf   T1XLo,W
008D 1C03  00260         btfss   STATUS,C
008E 28A5  00261         goto    CA1             ;Jump if TX1 < ZXactual
008F       00262 CA2
008F 080C  00263         movf    T1XHi,W         ;T1X - ZXactual
0090 0090  00264         movwf   AccHi
0091 080D  00265         movf    T1XLo,W
0092 0091  00266         movwf   AccLo
0093 0829  00267         movf    ZXActualHi,W
0094 008F  00268         movwf   ArgH
0095 082A  00269         movf    ZXActualLo,W
0096 008E  00270         movwf   ArgL
0097 210E  00271         call    Sub16x16
0098 3002  00272         movlw   KHi
0099 008F  00273         movwf   ArgH
009A 30D0  00274         movlw   KLo
009B 008E  00275         movwf   ArgL
009C 211A  00276         call    Mul16x16        ;PRODW3,PRODW2,PRODW1,PRODW0 =
           00277                                 ;K * (T1X - ZXactual)
009D 081B  00278         movf    T2Hi,W
009E 0098  00279         movwf   DIV1
009F 081C  00280         movf    T2Lo,W
00A0 0097  00281         movwf   DIV0
00A1 2165  00282         call    Div32x16        ;ANS1:ANS0=
00A2 0819  00283         movf    ANS0,W          ;  [K*(T1X-ZXactual)]/T2actual
00A3 00AD  00284         movwf   XAccel          ;The result will be a signed 8-bit #
00A4 28BB  00285         goto    DoYCalc
00A5       00286 CA1
00A5 0829  00287         movf    ZXActualHi,W    ;ZXactual - T1X
00A6 0090  00288         movwf   AccHi
00A7 082A  00289         movf    ZXActualLo,W
00A8 0091  00290         movwf   AccLo
00A9 080C  00291         movf    T1XHi,W
00AA 008F  00292         movwf   ArgH
```

```
00AB 080D  00293         movf    T1XLo,W
00AC 008E  00294         movwf   ArgL
00AD 210E  00295         call    Sub16x16
00AE 3002  00296         movlw   KHi
00AF 008F  00297         movwf   ArgH
00B0 30D0  00298         movlw   KLo
00B1 008E  00299         movwf   ArgL
00B2 211A  00300         call    Mul16x16        ;PRODW3,PRODW2,PRODW1,PRODW0 =
           00301                                 ;K * (ZXactual - T1X)
00B3 081B  00302         movf    T2Hi,W
00B4 0098  00303         movwf   DIV1
00B5 081C  00304         movf    T2Lo,W
00B6 0097  00305         movwf   DIV0
00B7 2165  00306         call    Div32x16        ;ANS1,ANS0 =
00B8 0919  00307         comf    ANS0,W          ;  [K*(ZXactual-T1X)]/T2actual
00B9 3E01  00308         addlw   0x01            ;The result will be a signed 8-bit #
00BA 00AD  00309         movwf   XAccel
00BB       00310 DoYCalc
00BB 082B  00311         movf    ZYActualHi,W    ;Check if acceleration is positive
00BC 0221  00312         subwf   T1YHi,W         ;or negative by comparing
00BD 1C03  00313         btfss   STATUS,C        ;T1Y and ZYactual
00BE 28DB  00314         goto    CA3             ;Jump if T1Y < ZYactual
00BF 1D03  00315         btfss   STATUS,Z        ;Test if T1YHI=ZY_ACTUAL_HI
00C0 28C5  00316         goto    CA4             ;Jump if T1Y > ZYactual
00C1 082C  00317         movf    ZYActualLo,W
00C2 0222  00318         subwf   T1YLo,W
00C3 1C03  00319         btfss   STATUS,C
00C4 28DB  00320         goto    CA3             ;Jump if TY1 < ZYactual
00C5       00321 CA4
00C5 0821  00322         movf    T1YHi,W         ;T1Y - ZYactual
00C6 0090  00323         movwf   AccHi
00C7 0822  00324         movf    T1YLo,W
00C8 0091  00325         movwf   AccLo
00C9 082B  00326         movf    ZYActualHi,W
00CA 008F  00327         movwf   ArgH
00CB 082C  00328         movf    ZYActualLo,W
00CC 008E  00329         movwf   ArgL
00CD 210E  00330         call    Sub16x16
00CE 3002  00331         movlw   KHi
00CF 008F  00332         movwf   ArgH
00D0 30D0  00333         movlw   KLo
00D1 008E  00334         movwf   ArgL
00D2 211A  00335         call    Mul16x16        ;PRODW3,PRODW2,PRODW1,PRODW0 =
           00336                                 ;K * (T1Y - ZYactual)
00D3 081B  00337         movf    T2Hi,W
00D4 0098  00338         movwf   DIV1
00D5 081C  00339         movf    T2Lo,W
00D6 0097  00340         movwf   DIV0
00D7 2165  00341         call    Div32x16        ;ANS1,ANS0 =
00D8 0819  00342         movf    ANS0,W          ;  [K*(T1Y-ZYactual)]/T2actual
00D9 00AE  00343         movwf   YAccel          ;The result will be a signed 8-bit #
00DA 0008  00344         return
00DB       00345 CA3
00DB 082B  00346         movf    ZYActualHi,W    ;ZYactual - T1Y
00DC 0090  00347         movwf   AccHi
00DD 082C  00348         movf    ZYActualLo,W
00DE 0091  00349         movwf   AccLo
00DF 0821  00350         movf    T1YHi,W
00E0 008F  00351         movwf   ArgH
00E1 0822  00352         movf    T1YLo,W
```

```
00E2 008E  00353         movwf   ArgL
00E3 210E  00354         call    Sub16x16
00E4 3002  00355         movlw   KHi
00E5 008F  00356         movwf   ArgH
00E6 30D0  00357         movlw   KLo
00E7 008E  00358         movwf   ArgL
00E8 211A  00359         call    Mul16x16        ;PRODW3,PRODW2,PRODW1,PRODW0 =
           00360                                 ;K * (ZYactual - T1Y)
00E9 081B  00361         movf    T2Hi,W
00EA 0098  00362         movwf   DIV1
00EB 081C  00363         movf    T2Lo,W
00EC 0097  00364         movwf   DIV0
00ED 2165  00365         call    Div32x16        ;ANS1,ANS0 =
00EE 0919  00366         comf    ANS0,W          ;[K*(ZYactual-T1Y)]/T2actual
00EF 3E01  00367         addlw   0x01            ;The result will be a signed 8-bit #
00F0 00AE  00368         movwf   YAccel
00F1 0008  00369         return
           00370
           00371
           00372 ;*****************************************************************
           00373 ;CheckCal
           00374 ;       This subroutine reads the CAL pushbutton switch (RB4) and if
           00375 ;       it is low, performs a simple calibration routine.
           00376 ;*****************************************************************
00F2       00377 CheckCal
00F2 1A06  00378         btfsc   PORTB,CAL       ;Is RB4 low?
00F3 0008  00379         return                  ;If not then exit routine
00F4 2276  00380         call    DisplayCal
00F5 2020  00381         call    ReadAccel       ;If yes then perform a read cycle
00F6 081B  00382         movf    T2Hi,W          ;Save the measured values in the
00F7 00A7  00383         movwf   T2calHi         ;calibration registers
00F8 081C  00384         movf    T2Lo,W
00F9 00A8  00385         movwf   T2calLo
00FA 080C  00386         movf    T1XHi,W
00FB 00A3  00387         movwf   ZXcalHi
00FC 080D  00388         movf    T1XLo,W
00FD 00A4  00389         movwf   ZXcalLo
00FE 0821  00390         movf    T1YHi,W
00FF 00A5  00391         movwf   ZYcalHi
0100 0822  00392         movf    T1YLo,W
0101 00A6  00393         movwf   ZYcalLo
0102 22B9  00394         call    WriteCal        ;Write the calibration data to EEPROM
0103 2292  00395         call    DisplayDone     ;Write message to LCD display
0104       00396 CCLoop
0104 1E06  00397         btfss   PORTB,CAL       ;Wait for pushbutton switch to be
0105 2904  00398         goto    CCLoop          ;released
0106 0008  00399         return
         00400 ;*****************************************************************
           00401
           00402
           00403 ;=================================================================
           00404 ;================== Mathematical Operations ======================
           00405 ;=================================================================
         00406 ;*****************************************************************
           00407 ;Add16x16
           00408 ;       This subroutine performs a 16-bit by 16-bit addition.
           00409 ;       Note that this routine does not check for possible overflow
           00410 ;       results i.e., 17-bit sum.
           00411 ;       Inputs are AccHi:AccLo and ArgH:ArgL
           00412 ;       Result is in AccHi:AccLo
```

```
                00413 ;               (AccHi:AccLo) = (AccHi:AccLo)+(ArgH:ArgL)
                00414 ;*****************************************************************
0107            00415 Add16x16
0107 080E       00416         movf    ArgL,W          ;Add low bytes together
0108 0791       00417         addwf   AccLo,F
0109 1803       00418         btfsc   STATUS,C        ;Check for carry out of addtion
010A 0A90       00419         incf    AccHi,F         ;If yes, increment AccHi
010B 080F       00420         movf    ArgH,W          ;Add high bytes together
010C 0790       00421         addwf   AccHi,F
010D 0008       00422         return
                00423
                00424 ;*****************************************************************
                00425 ;Sub16x16
                00426 ;       This subroutine performs a 16-bit by 16-bit subtraction.
                00427 ;       Inputs are AccHi:AccLo and ArgH:ArgL
                00428 ;       Result is in AccHi:AccLo
                00429 ;               (AccHi:AccLo) = (AccHi:AccLo)-(ArgH:ArgL)
                00430 ;*****************************************************************
010E            00431 Sub16x16
010E 098E       00432         comf    ArgL,F          ;2's complement ArgH:ArgL
010F 0A8E       00433         incf    ArgL,F
0110 1903       00434         btfsc   STATUS,2
0111 038F       00435         decf    ArgH,F
0112 098F       00436         comf    ArgH,F
0113 080E       00437         movf    ArgL,W          ;Now perform a 16-bit addition
0114 0791       00438         addwf   AccLo,F
0115 1803       00439         btfsc   STATUS,W
0116 0A90       00440         incf    AccHi,F
0117 080F       00441         movf    ArgH,W
0118 0790       00442         addwf   AccHi,F
0119 0008       00443         return
                00444
                00445 ;*****************************************************************
                00446 ;Mul16x16
                00447 ;       This subroutine performs a 16-bit by 16-bit multiplication.
                00448 ;       It produces a 32-bit number. Multiplication by 0 is checked
                00449 ;       and performed correctly, ie, A * 0 = 0.
                00450 ;       Inputs are (AccHi:AccLo) and (ArgH:ArgL)
                00451 ;       Output is (PRODW3:PRODW2:PRODW1:PRODW0)
                00452 ;         (PRODW3:PRODW2:PRODW1:PRODW0) = (AccHi:AccLo) * (Argh:ArgL)
                00453 ;*****************************************************************
011A            00454 Mul16x16
011A 01AF       00455         clrf    Temp0           ;Clear the temporary variables used
011B 01B0       00456         clrf    Temp1           ;in this routine
011C 01B1       00457         clrf    Temp2
011D 01B2       00458         clrf    Temp3
011E 0196       00459         clrf    PRODW0
011F 0195       00460         clrf    PRODW1
0120 0194       00461         clrf    PRODW2
0121 0193       00462         clrf    PRODW3
0122 0811       00463         movf    AccLo,W
0123 00AF       00464         movwf   Temp0           ;Move contents of AccHi:AccLo
0124 0810       00465         movf    AccHi,W         ;into Temp1:Temp0
0125 00B0       00466         movwf   Temp1
0126 0890       00467         movf    AccHi,F         ;Test if AccHi:AccLo = 0000
0127 1D03       00468         btfss   STATUS,Z
0128 292C       00469         goto    CheckNext       ;AccHi:AccLo not zero
0129 0891       00470         movf    AccLo,F
012A 1903       00471         btfsc   STATUS,Z
012B 2960       00472         goto    Equal0          ;AccHi:AccLo = 0000
```

```
012C       00473 CheckNext
012C 088F  00474         movf    ArgH,F          ;Test if ArgH:ArgL = 0000
012D 1D03  00475         btfss   STATUS,Z
012E 2932  00476         goto    DoMultiply      ;ArgH:ArgL not zero
012F 088E  00477         movf    ArgL,F
0130 1903  00478         btfsc   STATUS,Z
0131 2960  00479         goto    Equal0          ;ArgH:ArgL = 0000
0132       00480 DoMultiply
0132 088F  00481         movf    ArgH,F          ;Test if ArgH:ArgL has been reduced
0133 1D03  00482         btfss   STATUS,Z        ;to 0
0134 2938  00483         goto    TestLSB         ;ArgH:ArgL has not been reduced to 0
0135 088E  00484         movf    ArgL,F
0136 1903  00485         btfsc   STATUS,Z
0137 0008  00486         return                  ;ArgH:ArgL has been reduced to zero
           00487                                 ;so multiplication isdone
0138       00488 TestLSB
0138 1003  00489         bcf     STATUS,C        ;Shift ArgH:ArgL right
0139 0C8F  00490         rrf     ArgH,F
013A 0C8E  00491         rrf     ArgL,F
013B 1C03  00492         btfss   STATUS,C        ;Is LSb of ArgH:ArgL = 1
013C 295A  00493         goto    DoShift         ;Jump if LSb = 0
013D 082F  00494         movf    Temp0,W         ;If LSb = 1 then
013E 0796  00495         addwf   PRODW0,F        ;PRODW3:PRODW2:PRODW1:PRODW0 =
013F 1C03  00496         btfss   STATUS,C        ;PRODW3:PRODW2:PRODW1:PRODW0 +
0140 294A  00497         goto    ADD2            ;Temp3:Temp2:Temp1:Temp0
0141 3001  00498         movlw   0x01            ;Add carry bit if necessary
0142 0795  00499         addwf   PRODW1,F
0143 1C03  00500         btfss   STATUS,C
0144 294A  00501         goto    ADD2
0145 3001  00502         movlw   0x01            ;Add carry bit if PRODW1 overflows
0146 0794  00503         addwf   PRODW2,F        ;as a result of the addition of the
0147 1C03  00504         btfss   STATUS,C        ;previous carry
0148 294A  00505         goto    ADD2
0149 0A93  00506         incf    PRODW3,F
014A       00507 ADD2
014A 0830  00508         movf    Temp1,W
014B 0795  00509         addwf   PRODW1,F
014C 1C03  00510         btfss   STATUS,C
014D 2953  00511         goto    ADD3
014E 3001  00512         movlw   0x01
014F 0794  00513         addwf   PRODW2,F
0150 1C03  00514         btfss   STATUS,C
0151 2953  00515         goto    ADD3
0152 0A93  00516         incf    PRODW3,F
0153       00517 ADD3
0153 0831  00518         movf    Temp2,W
0154 0794  00519         addwf   PRODW2,F
0155 1C03  00520         btfss   STATUS,C
0156 2958  00521         goto    ADD4
0157 0A93  00522         incf    PRODW3,F
0158       00523 ADD4
0158 0832  00524         movf    Temp3,W
0159 0793  00525         addwf   PRODW3,F
015A       00526 DoShift
015A 1003  00527         bcf     STATUS,C        ;Shift temp registers left
015B 0DAF  00528         rlf     Temp0,F
015C 0DB0  00529         rlf     Temp1,F
015D 0DB1  00530         rlf     Temp2,F
015E 0DB2  00531         rlf     Temp3,F
015F 2932  00532         goto    DoMultiply
```

```
0160       00533 Equal0
0160 0196  00534         clrf    PRODW0          ;Since one arguement equals zero
0161 0195  00535         clrf    PRODW1          ;PRODW3,PRODW2,PRODW1,PRODW0 = 0
0162 0194  00536         clrf    PRODW2
0163 0193  00537         clrf    PRODW3
0164 0008  00538         return
           00539
           00540 ;*****************************************************************
           00541 ;Div32x16
           00542 ;       This subroutine performs a 32-bit x 16-bit division.
           00543 ;       Division is performed by binary long division.
           00544 ;       Inputs are (PRODW3:PRODW2:PRODW1:PRODW0) and (DIV1:DIV0).
           00545 ;       Output is (ANS1:ANS0)
           00546 ;          (ANS1:ANS0) = (PRODW3:PRODW2:PRODW1:PRODW0) / (DIV1:DIV0)
           00547 ;*****************************************************************
0165       00548 Div32x16
0165 019A  00549         clrf    ANS1            ;Clear the result registers
0166 0199  00550         clrf    ANS0
0167 3011  00551         movlw   0x11            ;DivCnt = 17d
0168 0092  00552         movwf   DivCnt
0169       00553 DA1
0169 0818  00554         movf    DIV1,W
016A 0213  00555         subwf   PRODW3,W        ;Is DIV1 > PRODW3
016B 1C03  00556         btfss   STATUS,C
016C 2973  00557         goto    NoSub           ;Jump if DIV1 > PRODW3
016D 1D03  00558         btfss   STATUS,2        ;Is DIV1 = PRODW3
016E 297C  00559         goto    DoSubs          ;Jump if DIV1 < PRODW3
016F 0817  00560         movf    DIV0,W          ;Is DIV0 > PRODW2
0170 0214  00561         subwf   PRODW2,W
0171 1803  00562         btfsc   STATUS,C
0172 297C  00563         goto    DoSubs          ;Jump if DIV0 < PRODW2
0173       00564 NoSub
0173 1003  00565         bcf     STATUS,C        ;Clear the carry bit
0174 0D99  00566         rlf     ANS0,F          ;Add 0 to LSb of ANS1,ANS0
0175 0D9A  00567         rlf     ANS1,F
0176 1003  00568         bcf     STATUS,C        ;Clear the carry bit
0177 0D96  00569         rlf     PRODW0,F        ;Shift PRODW3,2,1,0 left
0178 0D95  00570         rlf     PRODW1,F
0179 0D94  00571         rlf     PRODW2,F
017A 0D93  00572         rlf     PRODW3,F
017B 2991  00573         goto    ChkCnt
017C       00574 DoSubs
017C 0813  00575         movf    PRODW3,W
017D 0090  00576         movwf   AccHi
017E 0814  00577         movf    PRODW2,W
017F 0091  00578         movwf   AccLo
0180 0818  00579         movf    DIV1,W
0181 008F  00580         movwf   ArgH
0182 0817  00581         movf    DIV0,W
0183 008E  00582         movwf   ArgL
0184 210E  00583         call    Sub16x16        ;(PRODW3:2) = (PRODW3:2)-(DIV1:0)
0185 0810  00584         movf    AccHi,W
0186 0093  00585         movwf   PRODW3
0187 0811  00586         movf    AccLo,W
0188 0094  00587         movwf   PRODW2
0189 1403  00588         bsf     STATUS,C
018A 0D99  00589         rlf     ANS0,F
018B 0D9A  00590         rlf     ANS1,F          ;Add 1 to LSb of ANS1:ANS0
018C 1003  00591         bcf     STATUS,C
018D 0D96  00592         rlf     PRODW0,F        ;Shift PRODW3,2,1,0, left
```

```
018E 0D95  00593         rlf     PRODW1,F
018F 0D94  00594         rlf     PRODW2,F
0190 0D93  00595         rlf     PRODW3,F
0191       00596 ChkCnt
0191 0B92  00597         decfsz  DivCnt,F        ;Check for 17 operations
0192 2969  00598         goto    DA1                     ;If not then loop
0193 0008  00599         return
           00600 ;*****************************************************************
           00601
           00602
           00603 ;=================================================================
           00604 ;====================== Display Routines ==========================
           00605 ;=================================================================
           00606 ;*****************************************************************
           00607 ;DisplayAccel
           00608 ;       This subroutine takes the values int XAccel and YAccel and
           00609 ;       displays the ASCII equivalent on the LCD display.
           00610 ;*****************************************************************
0194       00611 DisplayAccel
0194 223E  00612         call    BusyXLCD        ;Wait for LCD to not be busy
0195 3001  00613         movlw   0x01            ;Reset cursor to home position
0196 221C  00614         call    WriteCmdXLCD    ;of line 1
           00615
0197 1FAD  00616         btfss   XAccel,7        ;Check if XAccel is negative
0198 29A0  00617         goto    XSpace
0199 223E  00618         call    BusyXLCD        ;Is negative
019A 302D  00619         movlw   '-'             ;Print a '-' to the display
019B 2254  00620         call    WriteDataXLCD
019C 092D  00621         comf    XAccel,W        ;2's complement XAccel
019D 3E01  00622         addlw   0x01
019E 00AD  00623         movwf   XAccel
019F 29A3  00624         goto    DispX
01A0       00625 XSpace                         ;Not negative
01A0 223E  00626         call    BusyXLCD
01A1 3020  00627         movlw   ' '             ;Print a space to the display
01A2 2254  00628         call    WriteDataXLCD
01A3       00629 DispX
01A3 082D  00630         movf    XAccel,W        ;Convert XAccel to 2-digit ASCII
01A4 22AC  00631         call    Bin2Ascii
01A5 223E  00632         call    BusyXLCD
01A6 0818  00633         movf    Digit1,W        ;Write the upper digit to the LCD
01A7 2254  00634         call    WriteDataXLCD
01A8 223E  00635         call    BusyXLCD
01A9 0817  00636         movf    Digit0,W        ;Write the lower digit to the LCD
01AA 2254  00637         call    WriteDataXLCD
01AB 223E  00638         call    BusyXLCD
01AC 30DF  00639         movlw   0xdf            ;Write a degrees symbol to the LCD
01AD 2254  00640         call    WriteDataXLCD
01AE 223E  00641         call    BusyXLCD
01AF 3020  00642         movlw   ' '             ;Write " Pit" to the LCD
01B0 2254  00643         call    WriteDataXLCD   ;for the word pitch which refers
01B1 223E  00644         call    BusyXLCD        ;to the X-axis
01B2 3050  00645         movlw   'P'
01B3 2254  00646         call    WriteDataXLCD
01B4 223E  00647         call    BusyXLCD
01B5 3069  00648         movlw   'i'
01B6 2254  00649         call    WriteDataXLCD
01B7 223E  00650         call    BusyXLCD
01B8 3074  00651         movlw   't'
01B9 2254  00652         call    WriteDataXLCD
```

```
01BA 223E  00653         call    BusyXLCD
01BB 30A8  00654         movlw   0xa8            ;Change the cursor position to home
01BC 221C  00655         call    WriteCmdXLCD    ;of line 2
           00656
01BD 1FAE  00657         btfss   YAccel,7        ;Check if YAccel is negative
01BE 29C6  00658         goto    YSpace
01BF 223E  00659         call    BusyXLCD        ;Is negative
01C0 302D  00660         movlw   '-'             ;Print a '-' to the display
01C1 2254  00661         call    WriteDataXLCD
01C2 092E  00662         comf    YAccel,W        ;2's complement YAccel
01C3 3E01  00663         addlw   0x01
01C4 00AE  00664         movwf   YAccel
01C5 29C9  00665         goto    DispY
01C6       00666 YSpace                         ;Not negative
01C6 223E  00667         call    BusyXLCD
01C7 3020  00668         movlw   ' '             ;Print a space to the display
01C8 2254  00669         call    WriteDataXLCD
01C9       00670 DispY
01C9 082E  00671         movf    YAccel,W        ;Convert YAccel to 2-digit ASCII
01CA 22AC  00672         call    Bin2Ascii
01CB 223E  00673         call    BusyXLCD
01CC 0818  00674         movf    Digit1,W        ;Write the upper digit to the LCD
01CD 2254  00675         call    WriteDataXLCD
01CE 223E  00676         call    BusyXLCD
01CF 0817  00677         movf    Digit0,W        ;Write the lower digit t the LCD
01D0 2254  00678         call    WriteDataXLCD
01D1 223E  00679         call    BusyXLCD
01D2 30DF  00680         movlw   0xdf            ;Write a degrees symbol to the LCD
01D3 2254  00681         call    WriteDataXLCD
01D4 223E  00682         call    BusyXLCD
01D5 3020  00683         movlw   ' '             ;Write " Rol" to the LCD
01D6 2254  00684         call    WriteDataXLCD   ;for the word roll which refers
01D7 223E  00685         call    BusyXLCD        ;to the Y-axis
01D8 3052  00686         movlw   'R'
01D9 2254  00687         call    WriteDataXLCD
01DA 223E  00688         call    BusyXLCD
01DB 306F  00689         movlw   'o'
01DC 2254  00690         call    WriteDataXLCD
01DD 223E  00691         call    BusyXLCD
01DE 306C  00692         movlw   'l'
01DF 2254  00693         call    WriteDataXLCD
01E0 0008  00694         return
           00695
           00696 ;*****************************************************************
           00697 ;OpenXLCD
           00698 ;       This subroutine initializes the LCD display. It is
           00699 ;       cleared and blank upon exit of this routine
           00700 ;*****************************************************************
01E1       00701 OpenXLCD
01E1 301E  00702         movlw   0x1e            ;Delay for POR
01E2 22A2  00703         call    Delay_Ms_4MHz
           00704
01E3 30F0  00705         movlw   0xf0            ;Write upper byte of configuration
01E4 1683  00706         bsf     STATUS,RP0      ;value to the LCD three times
01E5 0586  00707         andwf   TRISB,F         ;After this the LCD can be read
01E6 1283  00708         bcf     STATUS,RP0
01E7 0586  00709         andwf   PORTB,F
01E8 3003  00710         movlw   0x03
01E9 0486  00711         iorwf   PORTB,F         ;Output data to the port, 8-bit mode
01EA 1505  00712         bsf     PORTA,E         ;Clock the data in
```

```
01EB 0000  00713         nop
01EC 1105  00714         bcf     PORTA,E
           00715
01ED 300A  00716         movlw   0x0a            ;Wait for ~5ms
01EE 22A2  00717         call    Delay_Ms_4MHz
           00718
01EF 30F0  00719         movlw   0xf0
01F0 0586  00720         andwf   PORTB,F
01F1 3003  00721         movlw   0x03
01F2 0486  00722         iorwf   PORTB,F         ;Output data to the port, 8-bit mode
01F3 1505  00723         bsf     PORTA,E         ;Clock the data in
01F4 0000  00724         nop
01F5 1105  00725         bcf     PORTA,E
           00726
01F6 300A  00727         movlw   0x0a            ;Wait for ~5ms
01F7 22A2  00728         call    Delay_Ms_4MHz
           00729
01F8 30F0  00730         movlw   0xf0
01F9 0586  00731         andwf   PORTB,F
01FA 3003  00732         movlw   0x03
01FB 0486  00733         iorwf   PORTB,F         ;Output data to the port, 8-bit mode
01FC 1505  00734         bsf     PORTA,E         ;Clock the data in
01FD 0000  00735         nop
01FE 1105  00736         bcf     PORTA,E
           00737
01FF 30F0  00738         movlw   0xf0
0200 0586  00739         andwf   PORTB,F
0201 1486  00740         bsf     PORTB,1         ;Output data to the port, 4-bit mode
0202 1505  00741         bsf     PORTA,E
0203 0000  00742         nop
0204 1105  00743         bcf     PORTA,E
           00744
0205 300F  00745         movlw   0x0f
0206 1683  00746         bsf     STATUS,RP0
0207 0486  00747         iorwf   TRISB,F
0208 1283  00748         bcf     STATUS,RP0
           00749
0209 223E  00750         call    BusyXLCD        ;Function Set: 4-bit mode, 2 lines,
020A 302F  00751         movlw   0x2f            ;5x8 dots
020B 221C  00752         call    WriteCmdXLCD
           00753
020C 223E  00754         call    BusyXLCD        ;Display Cntrl: display, cursor off
020D 3008  00755         movlw   0x08
020E 221C  00756         call    WriteCmdXLCD
           00757
020F 223E  00758         call    BusyXLCD        ;Display Cntrl: display & cursor on,
0210 300F  00759         movlw   0x0f            ;blinking on
0211 221C  00760         call    WriteCmdXLCD
           00761
0212 223E  00762         call    BusyXLCD        ;Clear Display
0213 3001  00763         movlw   0x01
0214 221C  00764         call    WriteCmdXLCD
           00765
0215 223E  00766         call    BusyXLCD        ;Shift Cntrl: cursor moves to left
0216 3013  00767         movlw   0x13
0217 221C  00768         call    WriteCmdXLCD
           00769
0218 223E  00770         call    BusyXLCD        ;Set DDRAM address to 0
0219 3080  00771         movlw   0x80
021A 221C  00772         call    WriteCmdXLCD
```

```
021B 0008  00773         return
           00774
           00775
           00776 ;*****************************************************************
           00777 ;WriteCmdXLCD
           00778 ;       This subroutine writes a command to the LCD display using
           00779 ;       a 4-bit interface.
           00780 ;*****************************************************************
021C       00781 WriteCmdXLCD
021C 1283  00782         bcf     STATUS,RP0
021D 0099  00783         movwf   CMD             ;Save command in WREG to CMD
021E 30F0  00784         movlw   0xf0            ;Setup up data port for write
021F 1683  00785         bsf     STATUS,RP0
0220 0586  00786         andwf   TRISB,F
0221 1283  00787         bcf     STATUS,RP0
0222 0586  00788         andwf   PORTB,F
0223 0819  00789         movf    CMD,W           ;Write upper 4-bits to data port
0224 00AF  00790         movwf   Temp
0225 0EAF  00791         swapf   Temp,F
0226 300F  00792         movlw   0x0f
0227 052F  00793         andwf   Temp,W
0228 390F  00794         andlw   0x0f
0229 0486  00795         iorwf   PORTB,F
022A 1185  00796         bcf     PORTA,RW        ;Set the control bits for write
022B 1286  00797         bcf     PORTB,RS        ;and command
022C 0000  00798         nop
022D 1505  00799         bsf     PORTA,E         ;Clock the upper nibble in
022E 0000  00800         nop
022F 1105  00801         bcf     PORTA,E
0230 30F0  00802         movlw   0xf0
0231 0586  00803         andwf   PORTB,F
0232 300F  00804         movlw   0x0f
0233 0519  00805         andwf   CMD,W           ;Output the lower 4-bits to data port
0234 0486  00806         iorwf   PORTB,F
0235 0000  00807         nop
0236 1505  00808         bsf     PORTA,E         ;Clock the lower nibble in
0237 0000  00809         nop
0238 1105  00810         bcf     PORTA,E
0239 300F  00811         movlw   0x0f
023A 1683  00812         bsf     STATUS,RP0
023B 0486  00813         iorwf   TRISB,F
023C 1283  00814         bcf     STATUS,RP0
023D 0008  00815         return
           00816
           00817
           00818 ;*****************************************************************
           00819 ;BusyXLCD
           00820 ;       This subroutine monitors the busy bit from the LCD display
           00821 ;       It returns when the LCD is no longer busy.
           00822 ;*****************************************************************
023E       00823 BusyXLCD
023E 1283  00824         bcf     STATUS,RP0
023F 1585  00825         bsf     PORTA,RW        ;Set up for a read
0240 1286  00826         bcf     PORTB,RS        ;Read the busy bit/address
0241 0000  00827         nop
0242 1505  00828         bsf     PORTA,E         ;Clock the data out
0243 0000  00829         nop
0244 1D86  00830         btfss   PORTB,3         ;Read the busy bit
0245 2A4D  00831         goto    BNHI
0246 1105  00832         bcf     PORTA,E         ;Still busy
```

```
0247 0000  00833         nop
0248 1505  00834         bsf     PORTA,E         ;Clock out the lower nibble
0249 0000  00835         nop
024A 1105  00836         bcf     PORTA,E
024B 1185  00837         bcf     PORTA,RW
024C 2A3E  00838         goto    BusyXLCD        ;Try again
024D       00839 BNHI
024D 1105  00840         bcf     PORTA,E         ;LCD not busy
024E 0000  00841         nop
024F 1505  00842         bsf     PORTA,E         ;Clock out the lower nibble
0250 0000  00843         nop
0251 1105  00844         bcf     PORTA,E
0252 1185  00845         bcf     PORTA,RW
0253 0008  00846         return
           00847
           00848
           00849 ;*****************************************************************
           00850 ;WriteDataXLCD
           00851 ;       This subroutine writes a byte of data to the LCD display
           00852 ;       using the 4-bit interface.
           00853 ;*****************************************************************
0254       00854 WriteDataXLCD
0254 1283  00855         bcf     STATUS,RP0
0255 0099  00856         movwf   LDATA           ;Save the data in LDATA
0256 30F0  00857         movlw   0xf0            ;Setup the data port
0257 1683  00858         bsf     STATUS,RP0
0258 0586  00859         andwf   TRISB,F
0259 1283  00860         bcf     STATUS,RP0
025A 0586  00861         andwf   PORTB,F
025B 0819  00862         movf    LDATA,W         ;Write the upper nibble of data
025C 00AF  00863         movwf   Temp            ;to the data port
025D 0EAF  00864         swapf   Temp,F
025E 300F  00865         movlw   0x0f
025F 052F  00866         andwf   Temp,W
0260 390F  00867         andlw   0x0f
0261 0486  00868         iorwf   PORTB,F
0262 1686  00869         bsf     PORTB,RS        ;Set control signals for write
0263 1185  00870         bcf     PORTA,RW        ;to data registers
0264 0000  00871         nop
0265 1505  00872         bsf     PORTA,E         ;Clock the upper nibble in
0266 0000  00873         nop
0267 1105  00874         bcf     PORTA,E
0268 30F0  00875         movlw   0xf0
0269 0586  00876         andwf   PORTB,F
026A 300F  00877         movlw   0x0f
026B 0519  00878         andwf   LDATA,W         ;Write the lower nibble to data port
026C 0486  00879         iorwf   PORTB,F
026D 0000  00880         nop
026E 1505  00881         bsf     PORTA,E         ;Clock the lower nibble in
026F 0000  00882         nop
0270 1105  00883         bcf     PORTA,E
0271 300F  00884         movlw   0x0f
0272 1683  00885         bsf     STATUS,RP0
0273 0486  00886         iorwf   TRISB,F
0274 1283  00887         bcf     STATUS,RP0
0275 0008  00888         return
           00889
           00890 ;*****************************************************************
           00891 ;DisplayCal
           00892 ;       This subroutine displays a message to the LCD display
```

```
           00893 ;       indicating that a calibration cycle is in progress.
           00894 ;*****************************************************************
0276       00895 DisplayCal
0276 223E  00896         call    BusyXLCD
0277 3001  00897         movlw   0x01
0278 221C  00898         call    WriteCmdXLCD
0279 223E  00899         call    BusyXLCD
027A 3043  00900         movlw   'C'
027B 2254  00901         call    WriteDataXLCD
027C 223E  00902         call    BusyXLCD
027D 3061  00903         movlw   'a'
027E 2254  00904         call    WriteDataXLCD
027F 223E  00905         call    BusyXLCD
0280 306C  00906         movlw   'l'
0281 2254  00907         call    WriteDataXLCD
0282 223E  00908         call    BusyXLCD
0283 3069  00909         movlw   'i'
0284 2254  00910         call    WriteDataXLCD
0285 223E  00911         call    BusyXLCD
0286 3062  00912         movlw   'b'
0287 2254  00913         call    WriteDataXLCD
0288 223E  00914         call    BusyXLCD
0289 3072  00915         movlw   'r'
028A 2254  00916         call    WriteDataXLCD
028B 223E  00917         call    BusyXLCD
028C 3061  00918         movlw   'a'
028D 2254  00919         call    WriteDataXLCD
028E 223E  00920         call    BusyXLCD
028F 3074  00921         movlw   't'
0290 2254  00922         call    WriteDataXLCD
0291 0008  00923         return
           00924
           00925
           00926 ;*****************************************************************
           00927 ;DisplayDone
           00928 ;       This subroutine displays a message to the LCD display
           00929 ;       indicating that a calibration cycle has completed.
           00930 ;*****************************************************************
0292       00931 DisplayDone
0292 223E  00932         call    BusyXLCD
0293 30A8  00933         movlw   0xa8
0294 221C  00934         call    WriteCmdXLCD
0295 223E  00935         call    BusyXLCD
0296 3044  00936         movlw   'D'
0297 2254  00937         call    WriteDataXLCD
0298 223E  00938         call    BusyXLCD
0299 306F  00939         movlw   'o'
029A 2254  00940         call    WriteDataXLCD
029B 223E  00941         call    BusyXLCD
029C 306E  00942         movlw   'n'
029D 2254  00943         call    WriteDataXLCD
029E 223E  00944         call    BusyXLCD
029F 3065  00945         movlw   'e'
02A0 2254  00946         call    WriteDataXLCD
02A1 0008  00947         return
           00948
           00949
           00950
           00951 ;=================================================================
           00952 ;====================== Misc. Routines ===========================
```

```
                00953 ;=======================================================================
                00954 ;***********************************************************************
                00955 ;Delay_Ms_4MHz
                00956 ;       Generic delay routine.  Delay length in ms is loaded
                00957 ;       into WREG before calling.
                00958 ;***********************************************************************
02A2            00959 Delay_Ms_4MHz
02A2 1283       00960         bcf     STATUS,RP0
02A3 008E       00961         movwf   Count1
02A4            00962 DLMS2M1
02A4 307C       00963         movlw   0x7c
02A5 008F       00964         movwf   Count2
02A6            00965 DLMS2M2
02A6 0000       00966         nop
02A7 0B8F       00967         decfsz  Count2,F
02A8 2AA6       00968         goto    DLMS2M2
02A9 0B8E       00969         decfsz  Count1,F
02AA 2AA4       00970         goto    DLMS2M1
02AB 0008       00971         return
                00972
                00973
                00974 ;***********************************************************************
                00975 ;Bin2Ascii
                00976 ;       This routine converts a binary number to a 2-digit ASCII
                00977 ;       number.  The binary number is sent in WREG.
                00978 ;***********************************************************************
02AC            00979 Bin2Ascii
02AC 0198       00980         clrf    Digit1          ;Clear the upper digit
02AD 0097       00981         movwf   Digit0          ;Save the binary number
02AE            00982 B2A1
02AE 300A       00983         movlw   0x0a            ;Repeadedly subtract 10 from the
02AF 0217       00984         subwf   Digit0,W        ;number until the result is less
02B0 1C03       00985         btfss   STATUS,C        ;then 10
02B1 2AB5       00986         goto    B2A2
02B2 0097       00987         movwf   Digit0
02B3 0A98       00988         incf    Digit1,F
02B4 2AAE       00989         goto    B2A1
02B5            00990 B2A2
02B5 3030       00991         movlw   0x30            ;Add 0x30 to make the result
02B6 0797       00992         addwf   Digit0,F        ;ASCII
02B7 0798       00993         addwf   Digit1,F
02B8 3400       00994         retlw   0
                00995 ;***********************************************************************
                00996
                00997
                00998 ;=======================================================================
                00999 ;==================== Data EEPROM Routines ============================
                01000 ;=======================================================================
                01001 ;***********************************************************************
                01002 ;WriteCal
                01003 ;       This subroutine takes 6 bytes starting with address
                01004 ;       ZXcalHi and writes them to the internal Data EEPROM.
                01005 ;       Calls to WriteEE perform the actual write sequence.
                01006 ;***********************************************************************
02B9            01007 WriteCal
02B9 3006       01008         movlw   0x06            ;Load byte counter with 6
02BA 008E       01009         movwf   Count1
02BB 3023       01010         movlw   ZXcalHi         ;Load the starting address into FSR
02BC 0084       01011         movwf   FSR
02BD 01B4       01012         clrf    EADR            ;Start writing data to EE address 0
```

```
02BE       01013 WCLoop
02BE 0800  01014         movf    INDF,W          ;Load data
02BF 00B5  01015         movwf   EDATA
02C0 22D2  01016         call    WriteEE         ;Call routine to write data
02C1 0AB4  01017         incf    EADR,F          ;Increment EE address
02C2 0A84  01018         incf    FSR,F           ;Increment FSR
02C3 0B8E  01019         decfsz  Count1,F        ;Decrement count
02C4 2ABE  01020         goto    WCLoop
02C5 0008  01021         return
           01022
           01023
           01024 ;*****************************************************************
           01025 ;RestoreCal
           01026 ;       This subroutine reads 6 bytes from the Data EE starting
           01027 ;       with address 0 and saves them starting with ZXcalHi.
           01028 ;       Calls to ReadEE perform the actual read sequence.
           01029 ;*****************************************************************
02C6       01030 RestoreCal
02C6 3006  01031         movlw   0x06            ;Load byte counter
02C7 008E  01032         movwf   Count1
02C8 3023  01033         movlw   ZXcalHi         ;Load starting address into FSR
02C9 0084  01034         movwf   FSR
02CA 01B4  01035         clrf    EADR            ;Load starting EE address with 0
02CB       01036 RCLoop
02CB 22E4  01037         call    ReadEE          ;Read data from EE
02CC 0080  01038         movwf   INDF            ;Save in register
02CD 0AB4  01039         incf    EADR,F          ;Increment EE address
02CE 0A84  01040         incf    FSR,F           ;Increment FSR
02CF 0B8E  01041         decfsz  Count1,F        ;Decrement count
02D0 2ACB  01042         goto    RCLoop
02D1 0008  01043         return
           01044
           01045 ;*****************************************************************
           01046 ;WriteEE
           01047 ;       This is the subroutine to load the address and data into
           01048 ;       the special EE access registers and perform the EE write
           01049 ;       sequence.
           01050 ;*****************************************************************
02D2       01051 WriteEE
02D2 1283  01052         bcf     STATUS,RP0
02D3 0834  01053         movf    EADR,W          ;Load EE address
02D4 0089  01054         movwf   EEADR
02D5 0835  01055         movf    EDATA,W         ;Load EE data
02D6 0088  01056         movwf   EEDATA
02D7 1683  01057         bsf     STATUS,RP0
02D8 1208  01058         bcf     EECON1,EEIF
02D9 1508  01059         bsf     EECON1,WREN     ;EE write sequence
02DA 3055  01060         movlw   0x55            ;must be performed
02DB 0089  01061         movwf   EECON2          ;in this order
02DC 30AA  01062         movlw   0xaa            ;otherwise write
02DD 0089  01063         movwf   EECON2          ;does not take
02DE 1488  01064         bsf     EECON1,WR       ;place correctly
02DF       01065 eBusy
02DF 1E08  01066         btfss   EECON1,EEIF     ;Wait for write to complete
02E0 2ADF  01067         goto    eBusy
02E1 1108  01068         bcf     EECON1,WREN     ;Disable writes
02E2 1283  01069         bcf     STATUS,RP0
02E3 0008  01070         return
           01071
           01072
```

```
           01073 ;*******************************************************************
           01074 ;ReadEE
           01075 ;       This is the subroutine to read from the data EE using the
           01076 ;       special EE access registers.
           01077 ;*******************************************************************
02E4       01078 ReadEE
02E4 1283  01079         bcf     STATUS,RP0
02E5 0834  01080         movf    EADR,W          ;Load EE address
02E6 0089  01081         movwf   EEADR
02E7 1683  01082         bsf     STATUS,RP0
02E8 1408  01083         bsf     EECON1,RD       ;Perform the EE write sequence
02E9 1283  01084         bcf     STATUS,RP0
02EA 0808  01085         movf    EEDATA,W        ;Move data into WREG
02EB 0008  01086         return
           01087
           01088
           01089         end


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0200 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0240 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0280 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
02C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXX---- ----------------
2000 : -------X-------- ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   745
Program Memory Words Free:   279

Errors   :     0
Warnings :     0 reported,     0 suppressed
```

NOTES:

AN716

Migrating Designs from PIC16C74A/74B to PIC18C442

Author: *Brett Duane*
Microchip Technology Inc.

INTRODUCTION

The PIC18CXX2 was intended to make conversions from midrange controllers to enhanced controllers as easily as possible. Changes to register and bit names, and bit locations were kept to a minimum. The PIC18CXX2 was designed to be pin-compatible with 28-pin and 40-pin midrange microcontrollers.

This application note describes the minimum changes required to port code from the PIC16C74A to the PIC18C442, and are typical when migrating code from any midrange controller to any enhanced microcontroller.

Changes to the PIC16C74A code largely consists of renaming registers and bits, moving bits to new registers, and placing variables into the appropriate places in RAM. Where additional features have been added, changes to code have been suggested.

Note: Bits not defined in the PIC16C74A should not be modified in the PIC18C442 until the effects of these changes are known.

OSCILLATOR OPERATION

Changes to the oscillator circuit or device configuration with respect to the oscillator mode may be required. Oscillator performance should be verified to ensure that it starts and operates as expected.

Crystal oscillator modes may require changes in loading capacitors and/or oscillator mode. RC oscillators may operate at a different frequency than expected on the PIC18C442 (when using the same components).

Note 1: Even though compatible devices are tested to the same electrical specifications, the device characteristics may have changed due to changes in the manufacturing process. These differences should not affect systems that were designed well within the device specifications. For systems that operate close to or outside the specification limits, manufacturing differences may cause the device to behave differently.

Note 2: Oscillator operation should be verified to ensure that it starts and performs as expected. Adjusting the loading capacitor values and/or the oscillator mode may be required.

OSCILLATOR MODES

The PIC18C442 has more oscillator modes than the PIC16C74A. Two RC modes affect the use of one pin. OSC1 is used for the RC oscillator, as before. OSC2 may be used either for clock output ($F_{OSC}/4$) or as digital I/O pin RA6.

There are now two HS modes. The HS/PLL mode operates with a crystal or resonator from 4 MHz to 10 MHz while the HS mode can use a crystal or resonator up to 20 MHz.

At Power-On Reset, the OST (1024 oscillator cycles) and PWRT (optional 72 ms) delays occur as in the HS mode. When the PLL is enabled, an additional 2 ms delay is added to allow the Phase Locked Loop (PLL) to lock to the crystal frequency and stabilize.

HS/PLL mode uses a PLL to multiply the oscillator frequency by 4, providing an output up to 40 MHz to the instruction clock divider. This is divided to produce the Q clocks, resulting in the 10 MHz instruction clock rate.

When waking from SLEEP, the OST and 2 ms PLL delays are required for the oscillator and PLL to restart in HS/PLL mode.

CLOCK SWITCHING

The PIC18C442 now allows use of the Timer1 oscillator in place of the system oscillator. When operating from the Timer1 oscillator, the system oscillator is shut down as in SLEEP mode, and will require the same delays to restart as when waking from SLEEP. This allows continued operation at very low speed (F_{cycle} = 32 kHz/4 = 8 kHz), with power consumption almost as low as SLEEP mode. Clock switching is possible between any system oscillator mode and the Timer1 oscillator. However, the system oscillator mode cannot be changed.

Both of the following conditions must be met to allow clock switching:

- The Timer1 oscillator must be enabled by setting the T1OSCEN bit in T1CON<3>.
- Clock switching must be enabled by clearing the $\overline{\text{OSCSEN}}$ bit in CONFIG1H<5>.

The actual clock switching is performed by operating the SCS bit, OSCCON<0>. Clearing the SCS bit causes the controller oscillator to be used for the controller clock, while setting the SCS bit causes the Timer1 oscillator to be used.

Peripherals that depend on the system clock for timing will be affected by the change in clock frequency.

INSTRUCTION CHANGES

When migrating code from midrange to enhanced controllers, the user must become aware of changes to the instruction set and make appropriate changes to their code. Usually, this requires examining the points in the code where program execution branches, depending on the state of a STATUS bit. Sometimes new instructions can simplify existing code.

Appendix C lists instructions for which status bit operation has changed from the PIC16C74A to the PIC18C422. Table C-1 lists the instructions that are carried over from the PIC16C74A, but generally affect new status bits. Table C-2 lists the instructions that were not carried over, and provides replacement instructions. Table C-3 lists new instructions, the status bits they affect or detect, and a short description of what the instruction does.

RAM

In the PIC16C74A, all variables and Special Function Registers (SFR's) are stored in two banks. Each bank can provide up to 128 addresses. The first 32 locations in each bank are reserved for SFR's while the remainder is used for GPR's in RAM.

The PIC18C442 data memory is grouped into banks of 256 bytes each. All SFR's are contained in bank 15. The PIC18C442 can store all variables in bank 0 with the same addresses used in the PIC16C74A. Addresses should be 12-bits long, and have the form "0x0nn" (the first 4 bits are always 0 for bank 0, nn = address used in the PIC16C74A).

The BANKSEL directive and banking instructions of the PIC16C74A are no longer required in the PIC18C442, as all data variables can be stored in bank 0. References to the BANKSEL directive can be commented out, removed, or left in place "as is". As data memory requirements grow, either the BANKSEL directive or firmware can modify the BSR to select the correct bank.

1. Ensure that all RAM variables have 12-bit addresses assigned to them, and are located in bank 0.
2. When the code clears STATUS bits RP1 and RP0 for the first time, replace these lines with `clrf BSR`.
3. Comment out or remove all other references to RP0 or RP1.
4. Comment out or remove all references to the BANKSEL directive (optional).

Access banking is automatically used when an operand has a 12-bit address below 0x080 (GPRs in the bottom half of bank 0), or at and above 0xF80 (SFRs in the top half of bank 15). When accessing operands in the access bank range (0x000–0x07F or 0xF80–0xFFF), the BSR is ignored. Addresses in the lower half of the access range access bank 0, while addresses in the upper half of the access range access bank 15.

In the PIC16C74A, SFRs were contained in the lower 32 locations of each bank. In the PIC18C442, all SFRs are located in the top portion of bank 15, and have 12-bit addresses assigned to them (0xF80 to 0xFFF). When accessing the SFRs, their symbolic names should be used. Access banking will ignore the BSR and automatically select bank 15 to access SFRs. Most SFR names are unchanged or very similar to those in the PIC16C74A. (See Appendix A.)

The FSR register is replaced by the register pair FSR0H:FSR0L, and contains the entire 12-bit address required to access any SFR or GPR in any bank. Change FSR to FSR0L. FSR0H should be cleared. Use INDF0 as the operand in instructions to access the register or RAM selected by FSR0. The practice of using FSR0 to access SFR's is discouraged. Direct addressing using the SFR name uses less code and is easier to maintain.

1. If the FSR is used to access SFRs, replace the corresponding INDF operand with the SFR name.
2. Replace all `bsf STATUS,IRP` instructions and `bcf STATUS,IRP` instructions with `clrf FSR0H`.
3. Replace all occurrences of `BANKISEL varname` with `clrf FSR0H`.
4. Change FSR to FSR0L.
5. Change INDF to INDF0.

If FSR0L is incremented or decremented using `INCF` or `DECF` instructions beyond 8 bits, FSR0L will roll over, and the STATUS bits will be affected accordingly. FSR0H will not be affected by the rollover (see Example 1).

EXAMPLE 1: INCREMENTING FSR0L AND OVERFLOW

```
clrf   FSR0H     ; clear FSR0H
movlw  0xFF
movwf  FSR0L     ; load FSR0L with max
                 ; 8-bit count
incf   FSR0L, F  ; FSR0L incremented to 0x00
                 ; STATUS,C=1   STATUS,Z=1
                 ; FSR0H=0x00 (no change)
```


PROGRAM MEMORY

The PIC16C74A uses a 13-bit program counter to address program memory words. The PIC18C442 uses a 21-bit program counter to address program memory bytes and counts by two when fetching program instructions. Attempts to write a 1 to PCL<0> will result in PCL<0>=0. Therefore, the program counter will always access program memory with the address LSb always set to 0.

Writing to PCL will cause PCLATH and PCLATU to be written to PCH and PCU, respectively, as in the PIC16C74A. However, a read of PCL will update PCLATH and PCLATU from PCH and PCU.

The `CALL` and `GOTO` instructions contain all the addressing information required. Manipulation of PCLATH to prepare for jumps is not required, and will have no practical effect. Such code may be left in the program, commented out, or removed.

Instructions with $

Occasionally, programmers will use a '$' symbol to indicate the program address of the current opcode. '$' by itself still functions as before. However, since the program counter now addresses bytes instead of words, any offsets added to '$' need to be doubled to refer to the correct address (see Example 2).

RETLW Tables

Computed `GOTO` subroutines for `RETLW` tables will require modifications to the way the table offset is computed before the table is called, or modification of the offset within the table subroutine. Since a computed `goto` causes a jump to a `retlw` instruction, PCL <0> must be 0. This requires the offset to be doubled to jump to the correct location in the table.

Example 3 shows a way to modify the table subroutine to support a 256 entry table (offset = 0 to 255) without having to consider how the table was called, or possible code page boundary issues. One temporary RAM location and a label at the start of the table entries are required.

EXAMPLE 2: $ OFFSETS DOUBLED

```
 goto  $-6     ; replaces   goto  $-3
 goto  $+0x2E  ; replaces   goto  $+0x17
```

EXAMPLE 3: MODIFIED ROUTINE TO SUPPORT 256 ENTRY TABLE

```
        movf    OFFSET,w        ; OFFSET=0x00 to 0xFF
        call    table
        ...

        ORG     0x3C0
table   movwf   taboff          ; save table offset
        bcf     STATUS,C        ; clear STATUS bit
        rlcf    taboff,F        ; multiply by 2, save in taboff

        movlw   HIGH(tab_st1)   ; get high byte of table start
        btfsc   STATUS,C        ; test carry bit
        incf    WREG,W
        movwf   PCLATH          ; modify PCLATH if required

        movlw   LOW(tab_st1)    ; get low byte of table address
        addwf   taboff,W        ; add in offset
        btfsc   STATUS,C        ; test for overflow
        incf    PCLATH,F        ; increment if needed
        movwf   PCL             ; make jump, PCLATH and PCLATU are
                                : written to PCH and PCU

tab_st1 retlw   0x00            ; table body, first entry, offset=0
        retlw   0x01            ; (256 entry, 256 word/512 byte)
        ...
        ...
```

TABLES IN PROGRAM MEMORY

Computed GOTOs using RETLW tables can be performed on the PIC18C442, but this allows only one byte of data to be stored in each program memory instruction (16-bits), and limits the table size to 256 entries.

Table operations allow two 8-bit bytes of data to be stored in each program memory word, doubling table data density. There is no limit to the number of table entries, up to the maximum program memory. Program memory can also be read to calculate a program checksum to verify program integrity.

Table Reads

A 21 bit table pointer to program memory is loaded with the address of the data byte to be read. This pointer is stored in TBLPTRU<4:0>, TBLPTRH<7:0>, and TBLPTRL<7:0>. A TBLRD* instruction causes the data at that address to be placed into TABLAT where the program can use it as data.

TBLPTRU<4:0>, TBLPTRH<7:0>, and TBLPTRL<7:0> are SFRs in memory space. TBLPTRL<0> need not always be 0 as with PCL<0>. The TBLPTR group of registers can be automatically incremented or decremented using variations of the TBLRD* instruction. Table 1 shows the instructions and their effects on TABLAT and TBLPTR. Pointer increment/decrement operations affect all 21 bits of the TBLPTR registers.

TABLE 1: TABLE READ INSTRUCTIONS AND EFFECTS ON THE TABLE POINTER

Instruction	Effects
TBLRD*	Places copy of program memory byte in TABLAT
TBLRD*+	Places copy of program memory byte in TABLAT Increments TBLPTR after read
TBLRD*-	Places copy of program memory byte in TABLAT Decrements TBLPTR after read
TBLRD+*	Increments TBLPTR before read Places copy of program memory byte in TABLAT

Steps for Table Reads

The steps to perform a table read from program memory are:

1. Set the table pointer to the desired byte address. (TBLPTR may be even or odd as required.)
2. Execute a TBLRD instruction.
3. Read the byte retrieved from program memory in TABLAT.

Code for Table Reads

An example of table read code is shown in Example 4.

EXAMPLE 4: TABLE READ CODE

```
RdStr    movlw   HIGH(string)
         movwf   TBLPTRH                 ; load high byte of pointer (0x12)
         movlw   LOW(string)
         movwf   TBLPTRL                 ; load low byte of pointer (0x34)

read     tblrd*+                         ; read byte from program memory,
                                         ; and increment pointer one byte

         movff   TABLAT,PORTB            ; move byte from table latch to output port B

         tstfsz  TABLAT                  ; was retrieved byte a null?
         goto    read                    ; no, do loop again

         return


         ORG     0x1234
String   DW      "This is a test.",0x00 ; text string
```


INTERRUPTS

The PIC18C442 resets with the interrupt structure in a PIC16C74A compatible mode. The interrupt vector origin has been changed from 0x0004 in the PIC16C74A to 0x0008 in the PIC18C442.

Interrupt Pins

The RB0/INT pin on the PIC16C74A has been renamed to RB0/INT0. The PIC18C442 offers 2 additional interrupt pins. RB1 and RB2 have been renamed RB1/INT1 and RB2/INT2 to support the additional interrupt functions.

Interrupt Handling Registers

The INTCON register is mostly unchanged. Bits INTE and INTF are renamed INT0IE and INT0IF, respectively.

The INTCON2 register contains bits that determine which edge of INT0, INT1, and INT2 will trigger interrupts. Interrupt priority for Timer0 and PORTB Interrupt-on-change is set in INTCON2. PORTB weak pull-up resistors are controlled here.

The INTCON3 register contains INT1 and INT2 interrupt enable bits, the interrupt flag bits, and interrupt priority bits.

Interrupt flag bits located in PIR1 have interrupt enable bits in PIE1 and interrupt priority bits in IPR1. The same is true for PIR2, PIE2, and IPR2.

1. Change the interrupt vector origin from 0x0004 to 0x0008.
2. Rename INT bit to INT0, INTF to INT0IF, and INTE to INT0IE.
3. Change OPTION, NOT_RBPU to INTCON2, NOT_RBPU.
4. Change OPTION, INTEDG to INTCON2, INTEDG0.

INT0 is always a high priority interrupt. The interrupt enable bit, INT0IE, and interrupt flag bit, INT0IF, are located in the INTCON register.

With respect to interrupt control, all other register and bit names, and bit locations in the PIC16C74A are unchanged in the PIC18C442.

Interrupt Priority

The PIC18C442 also offers 2 levels of interrupt priority, each with its own interrupt vector. Interrupts assigned high priority take the high priority interrupt vector at 0x0008, while interrupts assigned low priority take the low priority interrupt vector at 0x0018. Interrupt priority is enabled by setting IPEN, RCON<7> (Interrupt Priority Enable).

When IPEN is clear, all interrupts are considered high priority and take the high priority vector. This is the priority mode compatible with the PIC16C74A.

When IPEN is set, interrupt priority is enabled. The functions of GIE, INTCON<7> and PEIE, INTCON<6> are modified. GIE becomes GIEH (Global Interrupt Enable High) and PEIE becomes GIEL (Global Interrupt Enable Low). If GIEL is clear, all low priority interrupts are disabled. If GIEH is clear, all high and low priority interrupts are disabled.

Each interrupt source has an associated interrupt priority bit to set its priority. When set, interrupt priority is high. When clear, interrupt priority is low. If an interrupt source has an interrupt flag bit in PIR1 or PIR2, corresponding interrupt priority bits will be located in IPR1 or IPR2. The remaining priority bits are located in INTCON2 and INTCON3.

The interrupt priorities for Timer0 and RBIF are set using TMR0IP, INTCON2<2> and RBIP, INTCON<2>. Interrupt priorities for the INT1 and INT2 pins are set using INT1IP, INTCON3<6> and INT2IP, INTCON3<7>. Clearing these bits will select low priority interrupts.

The interrupt enable bits for INT1 and INT2 are set using INT1IE, INTCON3<3>, and INT2IE, INTCON3<4>. The corresponding interrupt flag bits are INT1IF, INTCON3<3> and INT2IF, INTCON3<4>.

Return Address Stack

The PIC18C442 stack has a 31 level stack instead of the 8 level stack in the PIC16C74A. The PIC18C442 also allows access to the stack pointer, stack error bits, and top-of-stack contents. `PUSH` and `POP` instructions have been added to manipulate the stack contents and stack pointer

The PIC16C74A stack is 8 levels deep, and functions as a circular buffer. Pushes beyond the 8th push overwrite the 1st push, 2nd push, etc. Pops beyond the 1st push begin returning the 8th push, 7th push, etc. There is no indication of the state of the stack. The stack contents are not available to the program.

The PIC18C442 stack is 31 levels deep, and functions as a linear buffer. The 31st push will set the stack overflow status bit STKFUL, STKPTR<7>. The 32nd push will overwrite the 31st push. All pops beyond the 1st push return 0x0000, set the stack underflow status bit STKUNF, STKPTR<6>, and restarts the program (but does not reset the device). Optionally, a device reset can occur when the stack overflow and underflow status bits are set (see RESETs.) The STKFUL and STKUNF bits are reset only by a POR or by software. This allows the program to respond to stack errors.

When the controller is initialized, the stack pointer STKPTR contains 0x00, and points to a stack address that contains 0x00000, which is the RESET vector. A `PUSH` or `CALL` instruction, or an interrupt will increment the stack pointer to the next higher stack location to become the new top-of-stack where the PC is then stored. A return instruction (`RETURN` or `RETFIE`) will move the contents of the top-of-stack to the PC, and

decrement the stack pointer. The POP instruction simply decrements the stack pointer, discarding the contents of the top-of-stack.

The 21-bit top-of-stack can be accessed through the top-of-stack registers TOSU<4:0>, TOSH<7:0>, and TOSL<7:0>. The top-of-stack is readable and writable, allowing data to be stored and retrieved using the stack.

Fast Register Stack

The fast register stack is a group of registers that saves the contents of the WREG, STATUS, and BSR registers every time an interrupt or subroutine call occurs. This stack is one level deep and is not accessible to the user. When a return with the fast option (`retfie FAST`) is executed, the contents of the fast register stack are restored back to the WREG, STATUS, and BSR registers.

Calls may use the Fast Register Stack. A call with the fast stack option (`call label, FAST`) saves the WREG, STATUS, and BSR registers to the fast register stack. A corresponding return is required to restore these registers (`return FAST`).

If interrupts are enabled, the fast register stack cannot be used for a return from a call. If an interrupt occurs during a called subroutine, the contents of the fast register stack will be replaced by the current WREG, STATUS, and BSR contents at the time of the interrupt. After the interrupt returns, the fast register stack will still contain the WREG, STATUS, and BSR contents from when the interrupt was executed. If a subroutine should attempt to return using the fast register stack, an improper context will be restored.

If interrupt priority is enabled, only high priority interrupts can use the fast register stack. High priority interrupts may interrupt low priority interrupts at any time.

RESETS

The PIC18C442 responds to all the same RESET sources as the PIC16C74A.

The PCON register has been renamed to RCON. The $\overline{TO}$ and $\overline{PD}$ bits from the STATUS register have been moved to RCON. All bits retain the same functions in the PIC18C442 as they had in the PIC16C74A.

Power-On Reset

The PCON register of the PIC16C74A has been renamed RCON and contains the $\overline{POR}$ bit. Operation of the $\overline{POR}$ bit is unchanged.

Brown-Out Reset

The PCON register of the PIC16C74A has been renamed RCON and contains the $\overline{BOR}$ bit. Operation of the $\overline{BOR}$ bit is unchanged.

The Brown-out Reset (BOR) module can be configured as enabled or disabled in the PIC18C442 as in the PIC16C74A. When BOR is enabled, the Power-up Timer (PWRT) is also automatically enabled. The state of the PWRT enable bit, PWRTEN, CONFIG2L<0> is ignored. However, the PIC18C442 offers four BOR thresholds instead of one, and is selected using BORV1:BORV0, CONFIG2L<3:2>. The time that VDD must remain below VBOR (Parameter D005) has increased from the PIC16C74A (Parameter 35, TBOR).

1. Change PCON to RCON
2. Change STATUS, NOT_TO to RCON, NOT_TO
3. Change STATUS, NOT_PD to RCON, NOT_PD
4. Select V_{BOR} threshold in configuration (BORV1:BORV0, CONFIG2L<3:2>)

$\overline{MCLR}$

$\overline{MCLR}$ on the PIC18C442 operates the same as the PIC16C74A. No changes to the code or circuit are required.

WDT

In the PIC18C442, the WDT now has its own postscaler, independent of the Timer0 prescaler. The WDT is enabled when WDTEN, CONFIG2H<0> is set, disabled when clear. The WDT postscaler is programmed using WDTPS2:WDTPS0, CONFIG2H<3:1> to select a ratio from 1:1 to 1:128.

$\overline{TO}$, STATUS<4> and $\overline{PD}$, STATUS<3> bits have been moved to $\overline{TO}$, RCON<3> and $\overline{PD}$, RCON<2>. The operation of these bits is unchanged.

If the WDT has been disabled by clearing WDTEN, the WDT may be enabled under software control by setting SWDTE, WDTCON<0>, and disabled by clearing this bit. The WDT postscaler ratio can not be changed.

If the WDT is enabled using WDTEN, then changing SWDTE will have no effect.

Stack Over/Underflow

The PIC16C74A has an 8 level stack. Once the PC has been pushed to the stack 8 times, a 9th push would overwrite the 1st stack location without any errors being generated. The PIC18C442 uses a 31 level stack. When the stack is almost full (30 pushes), and another push occurs, the 31st push sets the STKFUL status bit. The 32nd push overwrites the 31st push.

Conversely, the stack is empty (all pushes have been popped) and another pop occurs, the STKUNF bit is set and the PC is loaded with the RESET vector address. This does not reset the controller, but does restart the code.

The device can be configured to perform a reset when either the STKFUL or STKUNF bits are set. Setting SVTREN, CONFIG4L<0> will allow the stack error bits to reset the controller. The STKFUL and STKUNF bits are cleared only by a POR or by software. A reset caused by a stack error will not clear these bits. To determine the cause of this reset, the user will have to poll the STKFUL and STKUNF bits and clear them when taking corrective action.

`RESET` Instruction

The PIC18C442 offers a `RESET` instruction. This instruction performs a device reset similar to a MCLR reset. All peripherals are reset and program execution resumes from the reset vector.

The RCON register contains the $\overline{RI}$ bit. The $\overline{RI}$ bit is set by POR, BOR, and WDT resets, and is cleared by the `RESET` instruction. The $\overline{TO}$, $\overline{PD}$, $\overline{BOR}$, and $\overline{POR}$ bits are unaffected. By polling the $\overline{RI}$ bit, RCON<4>, the reason for this reset can be determined.

TIMER0

The PIC18C442 Timer0 resets to a mode identical to the PIC16C74A. The Timer0 count is read and written using TMR0L instead of TMR0. The OPTION_REG register has been renamed T0CON.

The PSA bit, T0CON<3>, now only enables the Timer0 prescaler when clear, and disables the Timer0 prescaler when set (same effect as in the PIC16C74A with respect to Timer0).

If a different Timer0 prescaler ratio is required, PS2:PS0, OPTION<2:0> has been replaced by T0PS2:T0PS0, T0CON<2:0>, which functions identically with respect to Timer0.

Timer0 will set its interrupt flag bit (T0IF) when TMR0L overflows (same as when Timer0 overflows in the PIC16C74A).

The WDT and its postscaler are unaffected by settings in T0CON. See the section on the WDT.

The $\overline{RBPU}$ bit, OPTION<7>, has been renamed and moved to $\overline{RBPU}$, INTCON2<7>. INTEDG, OPTION<6> has been renamed and moved to INT0EDG, INTCON2<6>.

The changes required in code are:

1. Rename and move OPTION, NOT_RBPU to INTCON2, NOT_RBUP.
2. Rename and move OPTION_REG, INTEDG to INTCON2, INTEDG0.
3. Change OPTION_REG to T0CON.
4. Rename PS2:PS0 to T0PS2:T0PS0.
5. Operations modifying PS2:PS0 and PSA for the WDT are commented out. Modify CONFIG2H instead.
6. Timer0 reads/writes use TMR0L instead of TMR0.

Timer0 can operate as a 16-bit timer by clearing T08BIT, T0CON<6>. In this mode, TMR0H is written to the Timer0 high byte when TMR0L is written. TMR0H is updated from the Timer0 high byte when TMR0L is read. Timer0 interrupts will occur when Timer0 (in 16-bit mode) rolls over from 0xFFFF to 0x0000.

TIMER1

The PIC18C442 Timer1 module is upwardly compatible with the PIC16C74A Timer1 module.

When RD16, T1CON<7> is set, a read of TMR1L causes TMR1H to be updated from the Timer1 high byte. A write to TMR1L will cause the Timer1 high byte to be updated from TMR1H. In this mode, the user does not have to check to see if the low byte rolled over while reading the high byte, or stop the timer when loading it.

The Timer1 oscillator of the PIC18C442 is functionally identical to the PIC16C74A Timer1 oscillator. However, due to process changes, operation of the Timer1 oscillator should be verified to operate as expected.

Note:	Even though the user has made no changes to the Timer1 oscillator circuit, oscillator operation should be verified to ensure that it starts and performs as expected. Adjusting the loading capacitor values may be required.

TIMER2

The Timer2 module of the PIC18C442 is identical to the Timer2 module of the PIC16C74A. No code changes are required.

TIMER3

The PIC18C442 provides a fourth timer not present in the PIC16C74A. This timer is identical to Timer1 as implemented in the PIC18C442. Both timers can serve as a timebase for the CCP capture and compare functions, and may use the Timer1 oscillator. Both may be reset by the CCP compare special event trigger.

CAPTURE/COMPARE/PWM

The capture, compare, and PWM functions of the PIC16C74A are fully compatible with the PIC18C442. No code changes or circuit modifications are required.

The PIC18C442 CCP module offers an extra mode not present in the PIC16C74A. The compare mode can toggle the CCP output pin on a match.

A/D

The A/D module on the PIC18C442 resets to the same state as in the PIC16C74A. Code written for the PIC16C74A will run with only one change on the PIC18C442. Because the PIC18C442 has a 10-bit A/D module, two 8-bit registers are now required to make the 10-bit result available. ADRES has been renamed to ADRESH, and will contain the 8 MSb of the result. As long as the user treats the ADCON0 and ADCON1 registers as if they were part of the PIC16C74A, the A/D module will function the same as in the PIC16C74A.

A new register and three new bits in ADCON1 offer some enhanced features over the PIC16C74A. ADRESL holds the additional bits of the 10-bit conversion result. ADFM, ADCON1<7> controls the justification of the 10-bit result in ADRESH:ADRESL. If the user wishes to use an 8-bit result, clear bit ADFM (RESET state, compatible with the PIC16C74A), and the 8 MSbs are placed in ADRESH. The remaining 2 bits are stored in ADRESL<7:6>. Bits ADRESL<6:0> will be cleared. This format allows the user to use the 8-bit result in ADRESH in 8-bit math operations.

If the user wishes to make use of all 10 bits, set bit ADFM. The 8 LSb are stored in ADRESL<7:0> and the 2 MSbs of the result are stored in ADRESH<1:0>. Bits ADRESH<7:2> are cleared. This format is useful for taking the 10-bit result from ADRESH:ADRESL, and using it “as is” in 16-bit math operations.

The instruction clock can now run at 10 MHz (100 nS) when a 10 MHz oscillator drives the PLL. The clock sources available in ADCS1:ADCS0, ADCON0<7:6> do not allow TAD to be set to a minimum of 1.6 µSec with such high clock speeds. A third A/D clock select bit is provided in ADCS2, ADCON1<6>. When ADCS2 is set, the instruction clock is divided by 2 allowing TAD to be set correctly. The internal A/D RC oscillator is not affected.

The user can use an external voltage reference for the A/D conversion in the PIC16C74A and the PIC18C442. The PIC18C442 also allows the use of a low reference voltage. Depending on the setting of PCFG3:PCFG0, ADCON1<3:0>, the user can select as references the controller supply and ground, an external high reference and the controller ground, or external high and low references.

The references can not exceed the controller supply rails, but can modify the conversion range by introducing a positive offset, and reducing the full scale input voltage. Check the electrical specifications for limits on the reference voltages (Parameters A20, A20A, A21, A22, and A25).

Example 5 shows how to initialize the A/D module.

EXAMPLE 5: A/D SETUP

This example shows how to initialize the A/D module for the following conditions:

AN0 may be anywhere in the range of 0.5V to 3.5V. The input is currently at 2.296V. The controller is using an 8 MHz oscillator and the PLL. A 10-bit result is desired for use in 16-bit calculations.

The clock source is selected by setting ADCS2:ADCS0 to B“110”:

ADCS2 = B“1”, ADCS1:ADCS0 = B“10”.

The 32 MHz PLL output is divided by 64 to produce a 2.0 µSec TAD.

The port is configured to make AN0 and AN1 analog inputs, AN2 as the low reference (VREF-) and AN3 as the high reference (VREF+) by setting PCFG3:PCFG0 to B“1101”. 3.5V is applied to AN3, and 0.5V to AN2. These references will apply to all conversions on all channels.

The ADFM bit is set to select right justification of the result.

The AN0 channel is selected by setting CHS2:CHS0 to B“000”, and the conversion started by setting the GO/$\overline{\text{DONE}}$ bit, ADCON0<2>. When the GO/$\overline{\text{DONE}}$ bit clears, ADRESH:ADRESL will contain 0x0265.

USART

The PIC16C74A USART is upwardly compatible with the PIC18C442 USART. No code changes are required.

The USART can monitor the serial data in 9-bit mode. Setting the ADDEN, RCSTA<3> bit causes the USART to generate an interrupt only when the 9th received data bit is set, instead of every time a new data byte has been received. RCREG is unchanged until the 9th bit is set. When set, RCREG is loaded with the received address.

This is useful for networks that indicate that the other 8 bits are a device address by setting the 9th bit. When the 9th bit is clear, data is being sent.

SSP

SSPCON has been renamed SSPCON1. Otherwise, the PIC16C74A SSP module is upwardly compatible with the PIC18C442 MSSP module.

SPI Mode

The PIC16C74A SSP module is fully compatible with the PIC18C442 with respect to SPI mode. No changes to code are required.

I²C Mode

The PIC16C74A SSP module is upwardly compatible with the PIC18C442 MSSP module. No changes to code are required for conversion.

The SSP module used in the PIC16C74A provides limited support for master mode I²C. The MSSP module is used in the PIC18C442, and supports I²C master and multi-master modes in hardware.

Master mode has been added. SSPCON2 has been added to support hardware master modes. Slave mode now provides general call support.

Master Mode

The master SSP module (MSSP) supports master mode I²C in hardware through the use of SSPCON2. Arbitration for multi-master operation is provided. The baud rate generator is used to generate SCL.

In the PIC16C74A, master mode was implemented in software that monitored and controlled the SCL and SDA pins. Start (S) and stop (P) bit interrupts where provided by SSP hardware.

The PIC18C442 still supports such operation without code changes. However, master and multi-master modes are now provided by hardware.

When master mode is selected (SSPM3:SSPM0, SSPCON1<3:0> = B“1011”), and SSPEN, SSPCON1<5> is set, the SSP module will control SCL and SDA. Data and slave addresses (7-bit or 10-bit) with R/W bit are sent in SSPBUF, and the baud rate is set in SSPADD<6:0>.

Only one operation can be completed at a time. Each operation must complete before the next can be started. Successful completion is indicated by setting SSPIF when the SSP module is becomes idle. Operations are start, repeated start, stop, sending 1 byte of data, receiving 1 byte of data, and sending an ACK/NACK. If an attempt is made to program the next operation before the module becomes idle, the programming is ignored, the WCOL status bit, SSPCON1<7>, is set and SSPIF remains clear. WCOL is cleared by software. If the bus was busy, BCLIF, PIR2<3>, is set generating an interrupt.

Slave General Call Support

Slave general call support is enabled by setting GCEN, SSPCON2<7>. When an address match occurs (either slave address, or the general call address of 0x00 when GCEN is set), several actions occur.

- The received address is placed in the SSPBUF register
- The BUFFER FULL status bit BF, SSPSTAT<0>, is set
- An ACK pulse is generated
- An interrupt is generated by setting SSPIF, PIR1<3>

When the interrupt is serviced, SSPBUF must be read to determine if the interrupt was generated by a slave or general call address match.

When the slave is configured for 10-bit addresses with GCEN set, and the general call address is detected, the UA, SSPSTAT<1> bit will not be set. Instead, the slave will begin receiving data after the ACK is sent.

CONCLUSION

Conversion of a PIC16C74A application to run on a PIC18C442 consists of the following:

- Checking to make sure that the main and Timer1 (if used) oscillators work as expected
- Placing variables into bank 0 and assigning 12-bit addresses
- Modifying computed `goto` subroutines for reading tables
- Modifying the names and locations of bits and registers
- If the Brown-out Reset is enabled, select a BOR threshold

If desired, the user can make use of additional features offered by the PIC18C442. These are:

- Additional system oscillator modes
- System clock switching for reduced power requirements
- Additional memory (program and data)
- Data retrieval using program memory
- Additional interrupt pins
- Interrupt priority
- Stack access and status
- Optional stack error reset
- Fast register stack to save and restore context
- `RESET` instruction
- Timer0 operates as an 8 or 16-bit timer/counter
- A fourth timer, Timer3, that duplicates Timer1
- A 10-bit A/D module
- Full I²C master mode

APPENDIX A: CHANGED REGISTER/BIT LOCATIONS/NAMES

TABLE A-1: CHANGED REGISTER/BIT LOCATIONS/NAMES

PIC16C74A		PIC18C442		Notes
Register	**Bit**	**Register**	**Bit**	
OPTION_REG	NOT_RBPU	INTCON2	NOT_RBPU	
OPTION_REG	INTEDG	INTCON2	INTEDG0	
OPTION_REG	T0CS	T0CON	T0CS	
OPTION_REG	T0SE	T0CON	T0SE	
OPTION_REG	PSA	T0CON	PSA	WDT has own postscaler. Enabled using WDTEN, CONFIG2H<0>
OPTION_REG	PS2	T0CON	T0PS2	WDT postscaler set using WDTPS2, CONFIG2H<3>
OPTION_REG	PS1	T0CON	T0PS1	WDT postscaler set using WDTPS1, CONFIG2H<2>
OPTION_REG	PS0	T0CON	T0PS0	WDT postscaler set using WDTPS0, CONFIG2H<1>
PCON	NOT_POR	RCON	NOT_POR	
PCON	NOT_BOR	RCON	NOT_BOR	
STATUS	NOT_TO	RCON	NOT_TO	
STATUS	NOT_PD	RCON	NOT_PD	
INDF	—	INDF0	—	
FSR	—	FSR0L	—	
TMR0	—	TMR0L	—	
SSPCON	—	SSPCON1	—	
ADRES	—	ADRESH	—	

APPENDIX B: CODE CHANGES

```
;  PIC18C442 code       ;   replaced PIC16C74A code

   clrf    BSR          ;   bcf      STATUS,RP0
                        ;   bcf      STATUS,RP1
                        ;   (first occurrence only)
                        ;   (otherwise comment out or remove)

   movf    INDF0,w      ;   movf     INDF,w
   movf    TMR0L,w      ;   movf     TMR0,w
   movf    FSR0L,w      ;   movf     FSR,w
   movf    SSPCON1,w    ;   movf     SSPCON,w
   movf    ADRESH,w     ;   movf     ADRES,w

   movf    T0CON,w      ;   movf     OPTION_REG,w (TIMER0 OPERATIONS ONLY)
   movf    RCON,w       ;   movf     PCON,w
```


APPENDIX C: INSTRUCTION CHANGES FROM PIC16C74A TO PIC18C442

TABLE C-1: DIFFERENCES IN STATUS BIT OPERATION

Instruction	STATUS Bits		Notes
	16C74A	18C442	
ADDLW	C, DC, Z	C, DC, Z, **OV**, **N**	
ADDWF	C, DC, Z	C, DC, Z, **OV**, **N**	
ANDLW	Z	Z, **N**	
ANDWF	Z	Z, **N**	
COMF	Z	Z, **N**	
DECF	Z	**C**, **DC**, Z, **OV**, **N**	
INCF	Z	**C**, **DC**, Z, **OV**, **N**	
IORLW	Z	Z, **N**	
IORWF	Z	Z, **N**	
MOVF	Z	Z, **N**	
RETFIE	GIE	GIE/GIEH, PEIE/GIEL	Interrupt priority modifies names and functions of these bits
RLF	C, **DC**, Z	C, Z, **N**	Rotate left using carry bit
RRF	C, **DC**, Z	C, Z, **N**	Rotate right using carry bit
SUBLW	C, DC, Z	C, DC, Z, **OV**, **N**	
SUBWF	C, DC, Z	C, DC, Z, **OV**, **N**	
XORLW	Z	Z, **N**	
XORWF	Z	Z, **N**	

Legend: STATUS bits in **Bold** are affected in code conversion from PIC16C74A to PIC18C442

TABLE C-2: INSTRUCTIONS NO LONGER SUPPORTED

16C Instructions	16C STATUS Bits	Work-Around
CLRW	**Z**	Use CLRF WREG instead

Legend: STATUS bits in **Bold** are affected in code conversion from PIC16C74A to PIC18C442

TABLE C-3: NEW PIC18C442 INSTRUCTIONS

Instructions	STATUS Bits	Notes
ADDWFC	C, DC, Z, OV, N	Add WREG, F, and carry bit
BC		Conditional branch depending on carry STATUS bit
BN		Conditional branch depending on negative STATUS bit
BNC		Conditional branch depending on carry STATUS bit
BNN		Conditional branch depending on negative STATUS bit
BNOV		Conditional branch depending on overflow STATUS bit
BNZ		Conditional branch depending on zero STATUS bit
BOV		Conditional branch depending on overflow STATUS bit
BRA		Unconditional branch
BTG		Toggle bit b of file f
BZ		Conditional branch depending on zero STATUS bit
CPFSEQ		Branch depending on result of unsigned subtraction
CPFSGT		Branch depending on result of unsigned subtraction
CPFSLT		Branch depending on result of unsigned subtraction
DAW	C	Decimal Adjust WREG
DCFSNZ		Decrement file, skip next instruction if result is not zero
INFSNZ		Increment file, skip next instruction if result is not zero
LFSR		Move literal to file pointed to by FSR
MOVF	Z, N	Move file f
MOVFF		Move any file to any file
MOVLB		Move literal to BSR (Bank Select Register)
MULLW		Multiply literal with WREG
MULWF		Multiply WREG with file
NEGF	C, DC, Z, OV, N	2's complement, toggles sign of 8-bit signed data
POP		Discard top of stack, decrement stack pointer
PUSH		Increment stack pointer, copy PC to top of stack
RCALL		Call subroutine at offset
RESET	ALL	Reset controller
RLNCF	Z, N	Rotate left without using carry bit
RRNCF	Z, N	Rotate right without using carry bit
SETF		Set all bits of file
SUBFWB	C, DC, Z, OV, N	WREG - f with borrow (carry)
SUBWFB	C, DC, Z, OV, N	f - WREG with borrow (carry)
TBLRD		Read byte from table in program memory
TBLWT		Write byte to table in program memory
TSTFSZ		Test file, skip next instruction if file=0

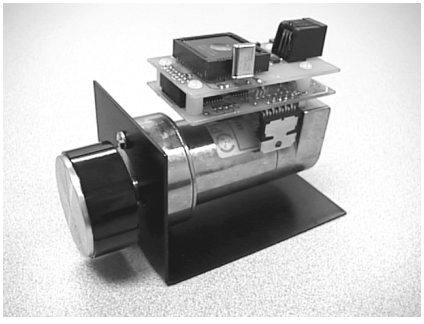
Brush-DC Servomotor Implementation using PIC17C756A

Author: *Stephen Bowling*
Microchip Technology Inc.

INTRODUCTION

This application note demonstrates the use of a PIC17C756A microcontroller (MCU) in a brush-DC servomotor application. The PIC17CXXX family of microcontrollers makes an excellent choice for cost-effective embedded servomotor control applications. Some of the benefits of the PIC17CXXX MCU family include fast instruction cycle execution (up to 120 ns), an 8 x 8 hardware multiplier, and many useful hardware peripherals. The application hardware is shown in Figure 1.

FIGURE 1: DC SERVOMOTOR APPLICATION HARDWARE



SYSTEM OVERVIEW

A block diagram of the servomotor system is provided in Figure 2. The system is comprised of the following elements:

- PIC17C756A MCU
- RS-232 Interface
- Power Amplifier
- Brush-DC Motor & Rotary Encoder

The MCU is responsible for communications with the host system, measuring the motor position, calculating the compensation algorithm and motion profile, and producing the drive signal sent to the power amplifier.

An RS-232 interface is the primary means of communication with the MCU. One of the two available USARTs on the MCU is used for this purpose. The operation of the motor is controlled and monitored from a host system using ASCII commands.

One of the three available pulse-width modulation (PWM) modules on the MCU is used to generate the motor drive signal. The PWM frequency is 32.2 kHz at a device operating frequency of 33 MHz and the module provides 10 bits of resolution. The torque applied to the motor is determined by the PWM duty cycle. The PWM signal is connected to a 'H'-bridge power amplifier capable of delivering up to 3A to the DC motor.

A Pittman Inc. 9234 series motor is used in this design. The motor has a no-load speed of 6151 RPM at 24 volts input and a torque constant of 5.17 oz-in/A (without gearbox). The peak stall current is 8.11A. A 5.9:1 ratio gearbox is installed on the output shaft.

A Hewlett Packard HEDS-9140 rotary optical encoder is mounted on the rear of the motor with a 500 count-per-revolution (CPR) encoder wheel mounted on the shaft. The encoder provides two pulse outputs that are in phase quadrature and a third index output that can be used to align the motor shaft to a reference position.

To save space, a stackable printed circuit board (PCB) system was designed that allows two PCBs to be mounted on top of the motor (see Figure 1). The bottom PCB contains a 5V regulator, motor driver, encoder interface, and limit switch buffer circuitry. The upper PCB contains the PIC17C756A MCU, crystal, RS-232 interface, and reset button.

HARDWARE DESCRIPTION

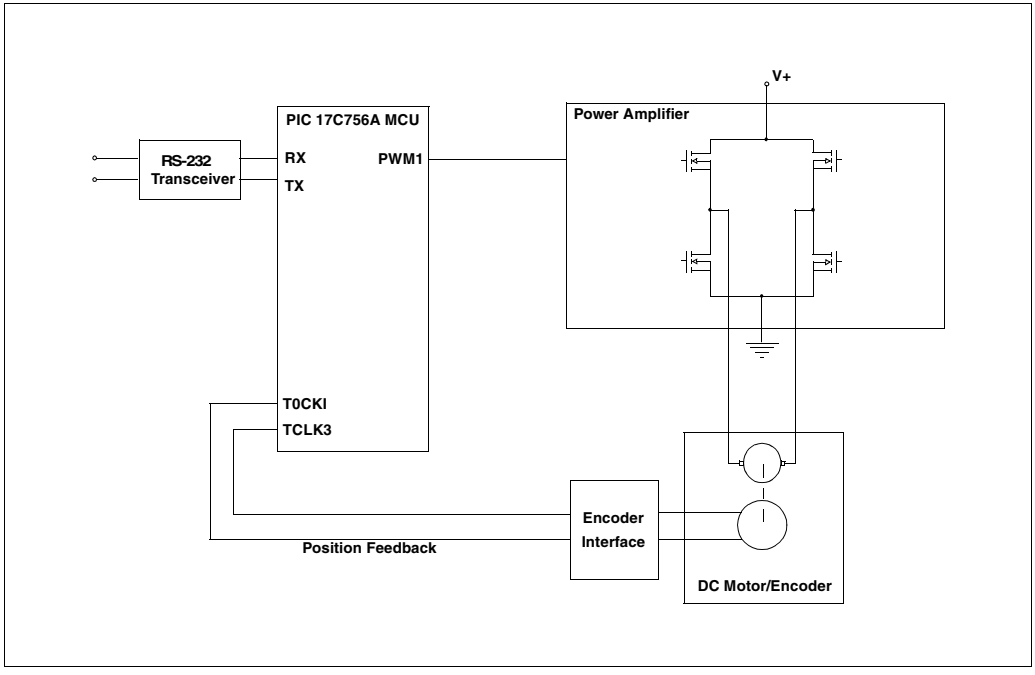
The design makes extensive use of the hardware peripherals available on the PIC17C756A. The peripherals used in this application are summarized in Table 1.

A complete schematic diagram for the application is given in Appendix A.

TABLE 1: PIC17C756A PERIPHERAL USAGE FOR DC SERVOMOTOR APPLICATION

Peripheral	Function
TMR0	Used as a counter to maintain the incremental up-count from the motor position encoder
TMR1	PWM1 time-base
TMR2	Servo update time-base
TMR3	Used as a counter to maintain the incremental down-count from the motor position encoder
PWM1	Generates drive signal for DC motor
USART1	Terminal communications
I/O	Encoder index signal, PWM amplifier enable, limit switch inputs

FIGURE 2: DC SERVOMOTOR BLOCK DIAGRAM



Motor Position Feedback

Referring to the schematic diagrams (Figure A-1 to Figure A-3), the outputs of the rotary encoder are connected to 2.7k pull-up resistors, filtered using RC networks, and buffered by Schmidt trigger inverters U5A - U5C. The outputs of the rotary encoder include two quadrature outputs and a third index output that is used to align the shaft of the motor to a known reference position. The conditioned index signal is connected to I/O pin RF0 of the MCU.

The conditioned quadrature outputs from the rotary encoder are connected to D flip-flops U6A and U6B. These D flip-flops decode the quadrature pulse train into up and down pulse outputs. A timing diagram indicating the operation of the decoder circuit is shown in Figure 3.

A simplified schematic diagram of the encoder interface is shown in Figure 4. The MCU accumulates the total distance traveled between servo updates based on the up and down pulse outputs from U6A and U6B. To accomplish this, Timer0 and Timer3 are configured as counters with external clock inputs. The output of D flip-flop U6A (up pulses) is connected to the Timer0 external clock input and the output of D flip-flop U6B (down pulses) is connected to the Timer3 external clock input. Each of these timer registers is 16 bits wide.

Three external logic inputs are provided at connector J4 on the motor driver PCB and are intended for mechanical limit switch sensing. These inputs could also be used to activate certain motor functions. The inputs are filtered and buffered by U5D – U5F similar to the encoder interface circuitry. The conditioned limit switch signals are connected to I/O pins RF1, RF2, and RF3 of the MCU.

PWM Amplifier

Integrated circuit U1 is an H-bridge driver that uses DMOS output devices and can deliver up to 3A output current at supply voltages up to 52V. The device has an internal charge pump for driving the high-side transistors and dead-time circuitry to prevent cross-conduction of the output devices. Each side of the bridge may be driven independently and the inputs are TTL compatible. An enable input and automatic thermal shutdown are also provided. A transient voltage suppressor is connected across the motor terminals to prevent voltage spikes generated by the motor inductance from damaging the bridge.

The PWM1 output from the MCU is buffered through inverters U3A, U3B, and U3D and connected to both sides of the H-bridge driver IC. One side of the bridge is driven with a inverted PWM signal. By driving the bridge in this manner, the motor may be turned in either direction depending on the PWM duty cycle. A 50% PWM duty cycle will produce zero motor torque. A 100% duty cycle will produce maximum motor torque in the forward direction, while a 0% duty cycle will produce maximum motor torque in the opposite direction.

An enable signal from I/O pin RF4 of the MCU is connected to the bridge driver through inverter U3C. This signal turns the output of the PWM amplifier on or off.

FIGURE 3: ENCODER TIMING

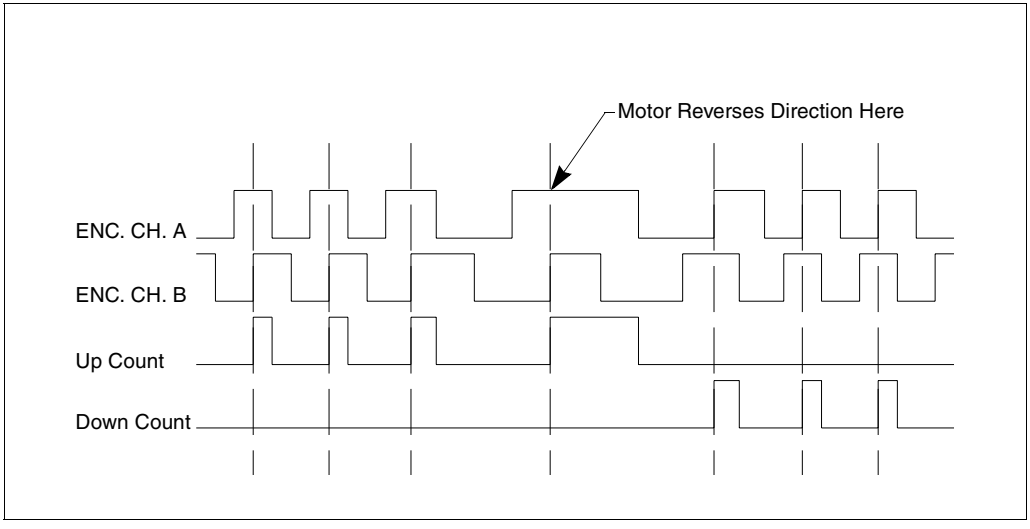
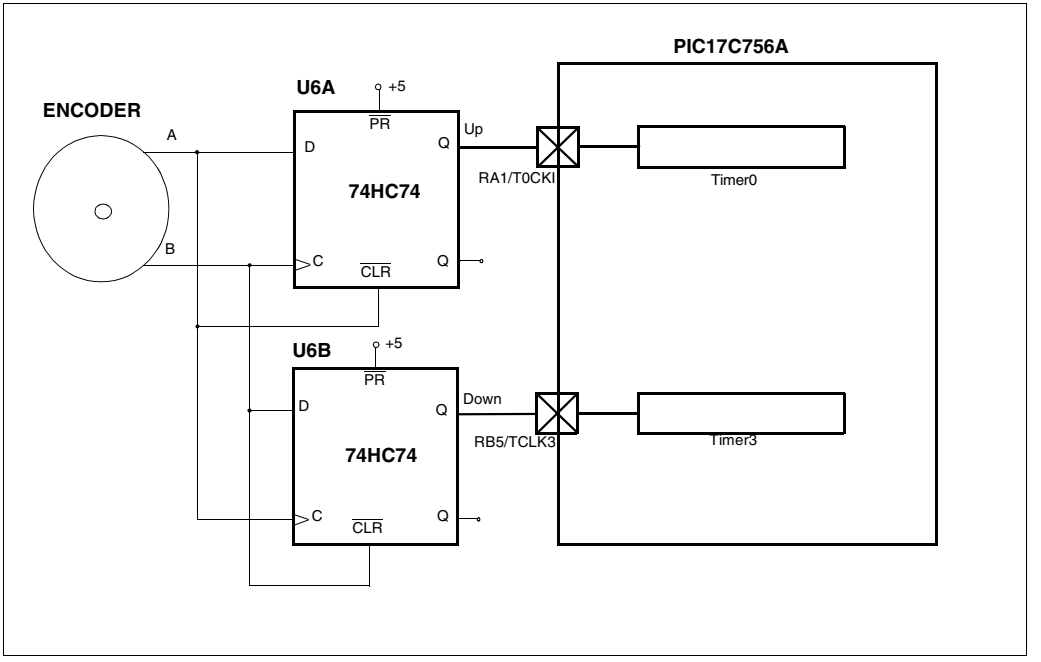


FIGURE 4: SIMPLIFIED ENCODER INTERFACE SCHEMATIC



Servo Update Timing

The servo update calculations are performed in an interrupt service routine and are synchronized with the output of PWM1. This is desirable because the duty cycle is updated at multiples of the PWM period. The PWM1 output is connected to the TCLK12/RB4 pin and is used as a clock source for Timer2. Timer2 has an associated period register, PR2. When the value of Timer2 is equal to the value loaded in PR2, Timer2 is reset to 0 and an interrupt is generated. By adjusting the value in PR2, the servo update frequency may be adjusted to any ratio of the PWM1 output. At a device operating frequency of 33 MHz, the frequency of PWM1 is 32.2 kHz. A 3.9 kHz servo update frequency will be achieved with the value in PR2 set to 8.

RS-232 Transceiver

The TX and RX pins of USART1 are connected to a Dallas Semiconductor DS275 RS-232 transceiver. The chip was selected for its small size and because it is line-powered. The chip uses power from the receive input to generate the correct RS-232 voltage levels while transmitting. To save space, RS-232 connections are made through a RJ-11 connector on the MCU PCB.

Power Supply

Voltage regulator VR1 provides 5 volts to the MCU, RS-232 driver, interface logic, and the rotary encoder. The system is designed to operate at any supply voltage between 10 volts and 24 volts. The supply voltage is connected directly to the PWM amplifier.

SOURCE CODE

The source code is written in the C programming language for ease of implementation and was compiled using the MPLAB™-C17 compiler. A complete source code listing for the application has been provided in Appendix B.

The source code performs four basic functions:

- RS-232 communication
- Motor position measurement
- Compensator algorithm calculation
- Motion profile calculation

All functions, except the RS-232 communications are performed in an interrupt service routine.

RS-232 Communications

The DC motor software allows control of the motor operating mode and parameter changes via a remote terminal with a RS-232 link operating at 19.2 kbaud. All RS-232 communication takes place in the main program loop. The USART1 reception interrupt flag (RC1IF) is polled to detect when a character has been received. Each received character is stored in a buffer, echoed to the USART, and the buffer index is incremented. This continues until the buffer is full or a <CR> is received. After a <CR> is received, the buffer contents are checked for numerical or command data and a 'READY>' prompt is sent to the terminal. If the command is not recognized, an error message is sent out.

Servo Updates

The servo calculations are performed each time a Timer2 interrupt occurs. A flowchart of the servo interrupt service routine (ISR) is shown in Figure 5.

32-bit Operations

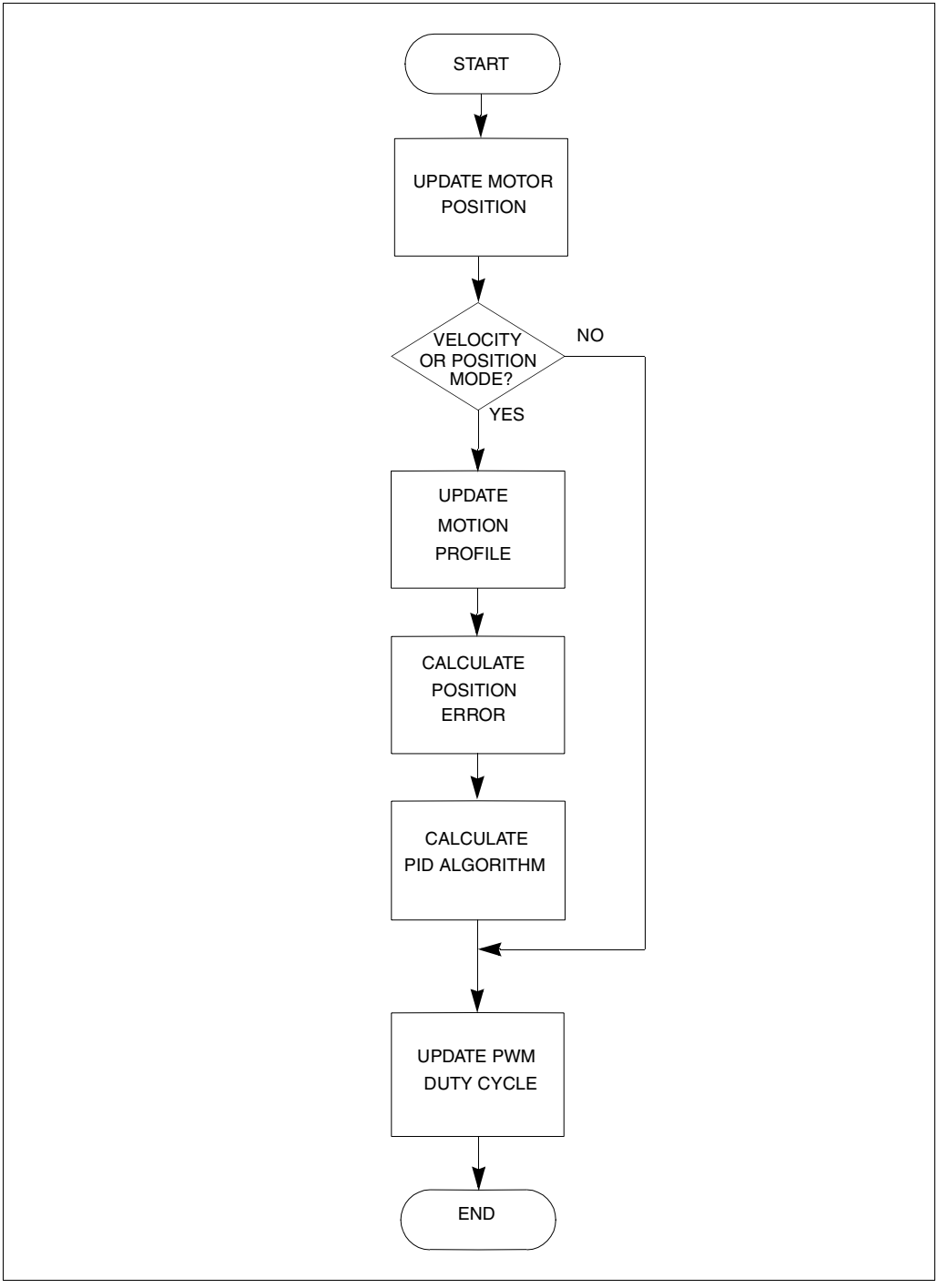
This application makes extensive use of 32-bit values. Since MPLAB-C17 does not provide direct support for 32-bit variable types, the 32-bit variables used in the program are declared as unions. The use of a union in the C programming language allows multiple variable types to share the same data space. A union with the name of 'LONG' has been declared in the source code. The union LONG consists of an array of four characters and an array of two integers. Therefore, any variables that are declared with this data type may be manipulated as four bytes or two integers. Additionally, the contents of the entire union may be copied to another location by simply assigning it to another union of the same type.

Position Updates

During each servo update period, the function `UpdatePosition()` is called. The count values in Timer0 and Timer3 are used to find the total motor distance traveled during the previous servo update period. The counters are never cleared to avoid the possibility of losing count information. Instead, the values of the Timer0 and Timer3 registers saved during the previous sample period are subtracted from the present values using two's-complement signed arithmetic. This calculation provides the total number of up and down pulses accumulated during the servo update period. The use of two's complement arithmetic accounts for a timer overflow that may have occurred since the last read. The down pulse count is then subtracted from the up pulse count, which provides a signed result indicating the total distance (and direction) traveled during the sample period. This value also represents the measured velocity of the motor in encoder counts per servo update period and is stored in the variable `mvelocity`.

The measured position of the motor is stored in the union `mposition`. The upper 24 bits of `mposition` holds the position of the motor in encoder counts. The lower eight bits of `mposition` represent fractional encoder counts. The value of `mvelocity` is added to `mposition` at each servo update period to find the new position of the motor. With 24 bits, the absolute position of the motor may be tracked through 33,554 shaft revolutions using a 500 CPR encoder. The size of `mposition` can be increased as necessary to track greater distances.

FIGURE 5: SERVO ISR FLOWCHART



The theoretical maximum encoder bit rate is determined by the number of bits in the counter registers and the servo update rate. If the counter should overflow between servo update periods, motor position information will be lost. A 16-bit counter register, for example, would provide $2^{16} - 1$ counts before an overflow occurred. Since two's complement arithmetic is used, the number of encoder counts during a given sample period must be limited to $2^{15} - 1$, or 32767. The maximum encoder rate is determined by multiplying the servo sampling frequency by the maximum encoder counts per sample. For this design, the servo update frequency is 3.9 kHz, which gives a theoretical maximum encoder rate of 128 MHz. In practice, the encoder rate is limited by the external clock timing specifications for Timer0 and Timer3. The minimum external clock period for Timer0 and Timer3 is T_{CY} + 40ns. Therefore, the maximum encoder rate is 6.2 MHz for a device operating frequency of 33 MHz.

PID Algorithm

The MCU must calculate and provide the correct motor drive signal based on the received motion commands and position/velocity feedback data. A compensation algorithm is used to ensure that the feedback loop is stabilized. Many types of algorithms may be used including various implementations of digital filters, fuzzy-logic, and the PID (proportional, integral, derivative) algorithm. A PID algorithm is used in this application since it is widely used in industrial applications and is easy to implement.

Figure 6 shows a flowchart indicating the function of the PID algorithm as it is implemented here. During each iteration of the servo loop, a position error is calculated and is used as the input to the algorithm. To control the operation of the PID algorithm, each of the three terms has a gain constant that can be adjusted in real-time by the user. Each term of the PID algorithm is calculated using a 16 bit x 16 bit signed multiplication algorithm with the PID gain constants `kp`, `ki`, and `kd` defined as 16-bit signed integers.

The union `position` holds the commanded motor position. The value of `mposition`, the measured motor position, is subtracted from `position` to find the present error in encoder counts. The least significant eight bits of these variables represent fractional encoder counts and are not used in the PID algorithm calculations. The `sub32()` function is used to subtract the values. The values to be subtracted are placed in `aarg` and `barg`. The result of the subtraction is available in `aarg` after the function has been called. The error calculation result in `aarg` is truncated to a signed 16-bit integer and stored in `u0`.

The multiplication routine is implemented as inline assembly instructions in the C source code. The algorithm executes in 36 cycles and takes advantage of the 8 x 8 hardware multiplier on the MCU. To perform the multiplication, the signed 16-bit integers to be multiplied are loaded into the `multplr` and `multcnd` variables and the function `mult()` is called. The 32-bit multiplication result is available in the union `aarg`. The `add32()` function is used to add the 32-bit terms of the PID algorithm.

The proportional term of the PID algorithm provides an output that is a function of the immediate position error, `u0`.

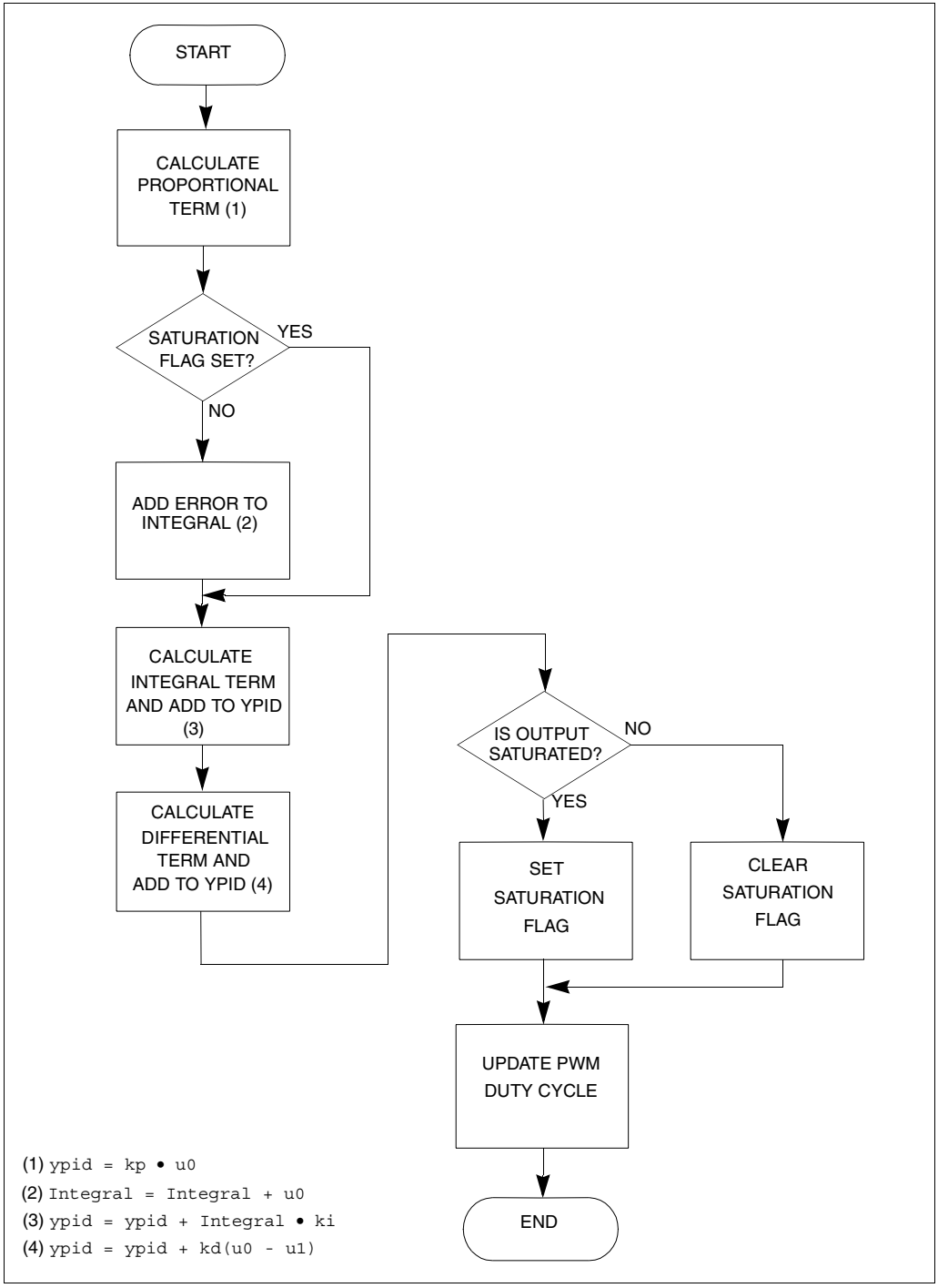
The integral term of the PID algorithm accumulates successive position errors calculated during each servo loop iteration and improves the low frequency open-loop gain of the servo system. The effect of the integral term is to reduce small steady-state position errors.

If the `stat.saturated` bit is set because the PWM output during the previous servo update period was saturated, the current position error is not be added to the integral value. This prevents a condition known as 'integrator-windup' that occurs when the integral term continues to accumulate error when the output is saturated. When the output is no longer saturated, the integral term 'unwinds' and causes abrupt motion as the accumulated error is reduced.

The differential term of the PID algorithm is a function of the difference in error between the current servo update period and the previous one. The integral term improves the high frequency open-loop response of the servo system.

After the three terms of the PID algorithm are summed, the 32-bit result stored in `ypid` is saturated to 24 bits. The 16-bit signed integer `ypwm` is used to set the PWM duty cycle. The upper 16 bits of `ypid` are used to set the duty cycle, which effectively divides the output of the PID algorithm by 256. The range of the duty cycle is restricted so that the PWM duty cycle cannot be less than 1% or greater than 99%. This ensures that Timer2 will always receive a valid clock input for the servo update timing interrupt. If beyond the limits, `ypwm` is set to the maximum allowable positive or negative value and `stat.saturated` is set to '1'. An offset value of 512 must be added to `ypwm` before it is written to the PWM duty cycle registers. (For 10-bit PWM resolution, a value of '0' written to the duty cycle registers provides a 0% duty cycle and a value of 1023 provides a 100% duty cycle.)

FIGURE 6: PID ALGORITHM FLOWCHART



Motion Profile

For optimum motion control, a method must be implemented that will control the motor acceleration and deceleration. Motion will be abrupt without the profile, causing excessive wear on the mechanical components and degrading the performance of the compensation algorithm.

For this application, a simple motion profile that generates trapezoidal (or triangular) moves has been implemented. The profile characteristics are adjusted by specifying a 16-bit velocity limit, `vlim`, and a 16-bit acceleration value, `accel`. The motion profile is used in Velocity Mode and Position Mode. If the motor is operating in one of these modes, the function `UpdateTrajectory()` is called each time `ServoISR()` is executed.

A specific motor velocity is established by adding an offset value to the commanded position at each servo update period. The 32-bit variable `velact` is used in the profile to hold the present commanded velocity of the motor. The lower 24 bits of `velact` and the least significant 8 bits of `position`, the commanded motor position, represent fractional encoder counts. The purpose of these additional bits is to increase the range of velocities that may be achieved. To achieve a particular motor velocity, the upper 16 bits of `velact` are added to `position` during each step of the profile. This allows the commanded motor velocity to vary between 1/256 counts/T_S and 127 counts/T_S. The actual velocity range of the motor is dependent on the servo update rate and the resolution of the encoder. With a 3.9 kHz servo update rate and a 500 CPR encoder, the range of commanded motor velocities is from 1.8 RPM to 59,436 RPM.

Motor acceleration/deceleration is accomplished in a manner similar to the motor velocity. The value of `accel` is added to or subtracted from `velact` at each servo update period.

A flowchart for the operation of the motion profile in Velocity Mode is shown in Figure 7. In Velocity Mode, data entered at the prompt is stored in the commanded velocity variable, `velcom`. After `velcom` is updated, the motor begins to accelerate or decelerate to the new commanded velocity. Acceleration continues until `velact` is equal to `velcom` or the velocity limit, `vlim`, has been exceeded. The value of `velact` is added to the commanded motor position, `position`. The motor will continue to run at the commanded velocity or the velocity limit until further velocity data is received. If the output is saturated (`stat.saturated = '1'`) during a particular servo update period, the commanded position is not changed.

A flowchart for the operation of the motion profile in Position Mode is shown in Figure 8. In Position Mode, a 16-bit relative movement distance is entered as encoder counts divided by 256. The total movement distance is divided by 2 and placed in `phase1dist`. A second variable, `flatcount`, is set to zero. The direction of the move is determined and stored in the `stat.neg_move` flag. The final move destination is calculated based on the present measured position and is stored in `fposition`. Finally, the `stat.move_in_progress` flag is set. Further position commands are ignored until the move has completed and this flag is cleared.

The motor begins to accelerate and the value of `velact` is subtracted from `phase1dist` at each servo update period to keep track of the distance traveled in the first half of the move. The value of `velact` is added or subtracted from the commanded motor position, `position`, depending on the state of the `stat.neg_move` flag. The motor stops accelerating when `velact` is greater than `vlim`. After the velocity limit has been reached, `flatcount` is incremented at each servo update period to keep track of the time spent in the flat portion of the move.

The first half of the move is completed when `phase1dist` becomes negative. At this time, the `stat.phase` flag is set to '1'. The variable `flatcount` is then decremented at each servo period. When `flatcount = 0`, the motor begins to decelerate. The move is complete when `velact = 0`. The previously calculated destination in `fposition` is written to the commanded motor position and the `stat.move_in_progress` flag is cleared at this time.

FIGURE 7: MOTION PROFILE FLOWCHART – VELOCITY MODE

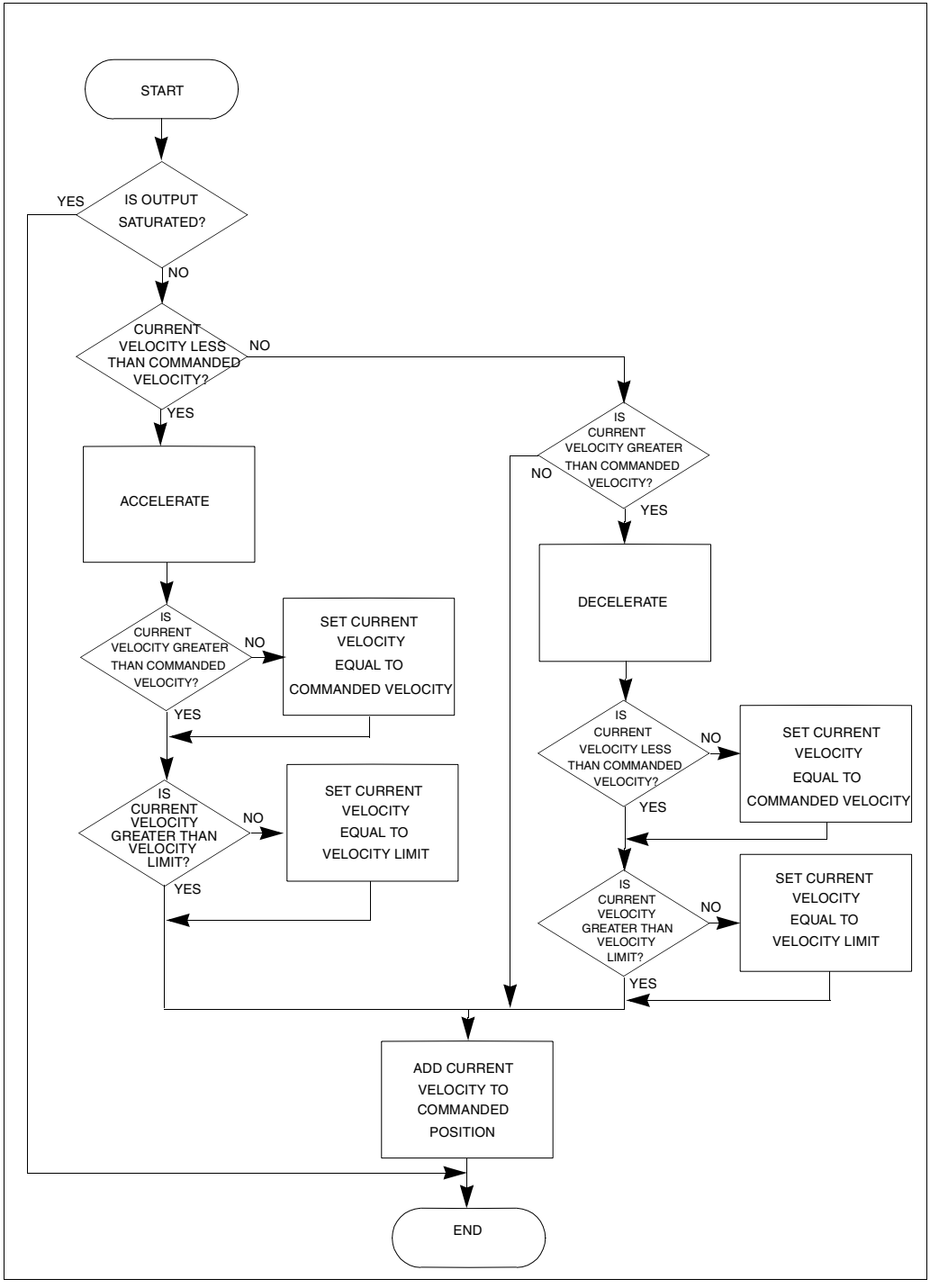
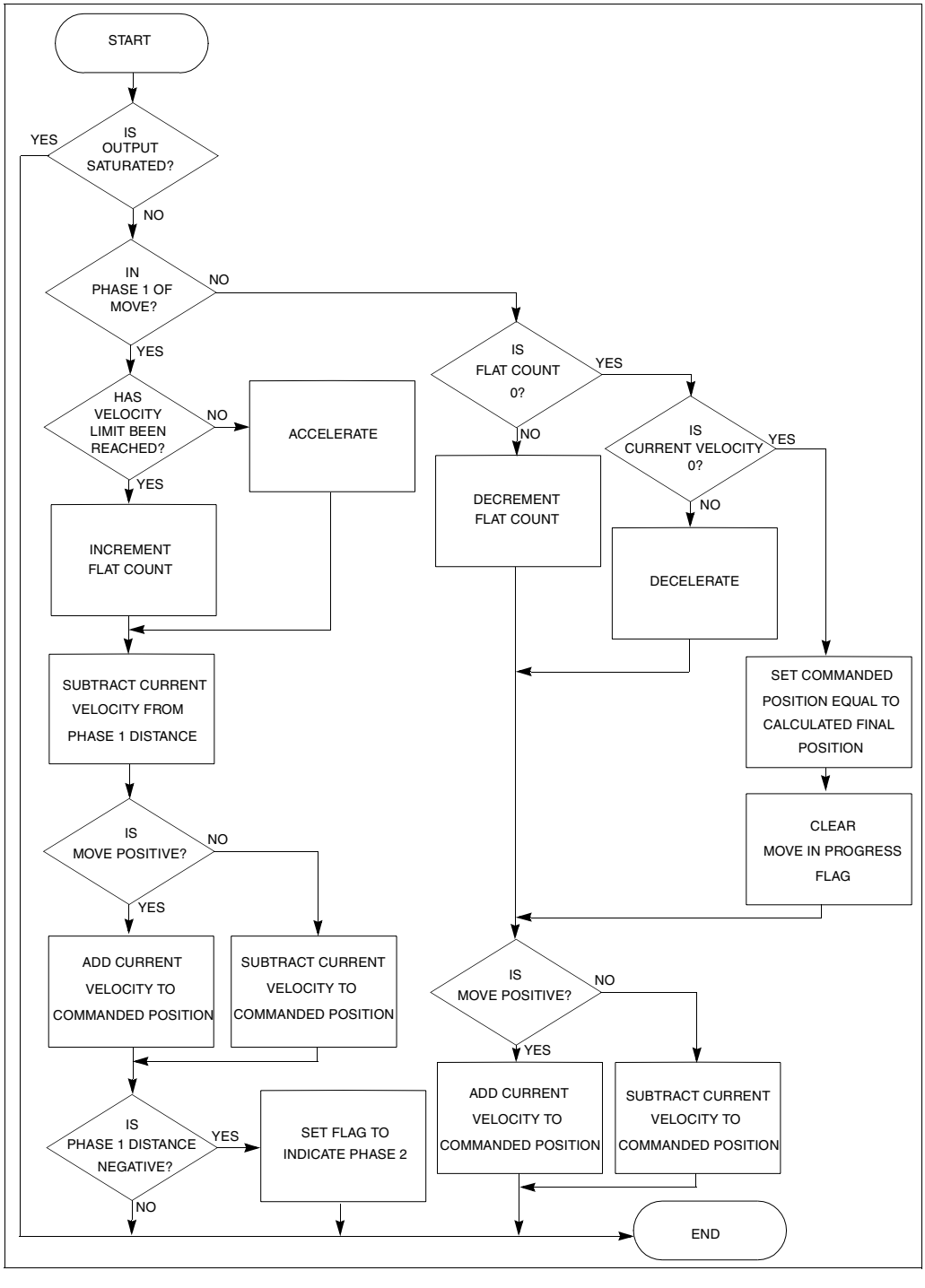


FIGURE 8: MOTION PROFILE FLOWCHART – POSITION MODE



USER INTERFACE

When power is first applied to the motor, the user will see a 'READY>' prompt appear on the terminal. At this time, the DC motor is ready to receive commands. A summary of all the commands is given in Table 2.

The software that controls the DC motor allows three basic modes of operation that are selectable from the remote terminal. These modes include Manual Mode, Velocity Mode, and Position Mode.

The default mode for the motor at power-up is Manual Mode. No position feedback is used in Manual Mode. The data entered at the prompt directly controls the PWM duty cycle delivered to the motor.

In Velocity Mode, the entry data specifies the signed motor velocity, which is given as encoder counts per sample period multiplied by 256. When new velocity data has been entered, the motor will accelerate or decelerate to the new velocity at a rate specified by the acceleration value. The motor will not accelerate if the velocity limit has been reached.

In Position Mode, the entry data specifies a signed 16-bit relative move distance. The movement distance, entered at the prompt, is given as encoder counts divided by 256. When a move distance is specified, a motion status flag is set and any additional move data are ignored until the current move is complete.

The profile of the move will be trapezoidal or triangular depending on the total move distance, the velocity limit, and the acceleration value. For a trapezoidal move, the motor will accelerate to the velocity limit and remain at that velocity until it is time for the motor to decelerate. If half of the move distance has been traveled before the motor reaches the velocity limit, the motor will begin to decelerate and the move will be triangular.

The motor operating parameters are displayed using the 'R' command. Any of the parameters may be modified by first entering the command to change the parameter, followed by a carriage return (<CR>). The parameter is then modified by entering the new value followed by a <CR>. The user can then verify that the parameter was changed by using the 'R' command again.

SUMMARY

The use of the PIC17C756A MCU in a DC servomotor application has many features that allow a cost-effective implementation with few external components. These include (2) 16-bit counters for position measurement, hardware PWM modules, and a hardware multiplier for high computational throughput.

`ServoISR()`, as written for this application, executes in 780 instruction cycles. For a servo update rate of 3.9kHz and a MCU clock frequency of 33 MHz, only 37% of the total MCU processing time is consumed. This provides additional time for performing unrelated tasks, computing more complicated compensator algorithms, or increasing the servo update rate.

TABLE 2: DC SERVO MOTOR COMMAND SUMMARY

Command	Data Range	Description
M <CR>	-500 ≤ data ≤ 500	Changes to the manual mode of operation. All subsequent data input is written directly to the PWM output.
V <CR>	-32768 ≤ data ≤ 32767	Changes to velocity mode. All subsequent data input is velocity in encoder counts per sample period multiplied by 256.
P <CR>	-32768 ≤ data ≤ 32767	Changes to position mode. All subsequent data input is a relative position move in encoder counts multiplied by 256.
W <CR>		Enables/disables PWM drive to the motor; the default is disabled.
R <CR>		Displays current K_P, K_I, K_D, velocity limit, and acceleration limit.
L <CR>		Displays the present motor position in hexadecimal format.
KP <CR> data <CR>	-32768 ≤ data ≤ 32767	Changes the proportional gain factor of the PID algorithm. The command is followed by the data value.
KI <CR> data <CR>	-32768 ≤ data ≤ 32767	Changes the integral gain factor of the PID algorithm. The command is followed by the data value.
KD <CR> data <CR>	-32768 ≤ data ≤ 32767	Changes the differential gain factor of the PID algorithm. The command is followed by the data value.
KV <CR> data <CR>	0 ≤ data ≤ 65535	Changes the velocity limit of the trajectory profile. The data value is encoder counts per sample period multiplied by 256. The command is followed by the data value.
KA <CR> data <CR>	0 ≤ data ≤ 65535	Changes the acceleration value for the trajectory profile. The command is followed by the data value.
KS <CR> data <CR>		Changes the servo update rate. The data value is written to the period register for Timer2. The servo update rate will be the PWM frequency divided by the value entered here.

APPENDIX A: SCHEMATICS

FIGURE A-1: SCHEMATIC 1

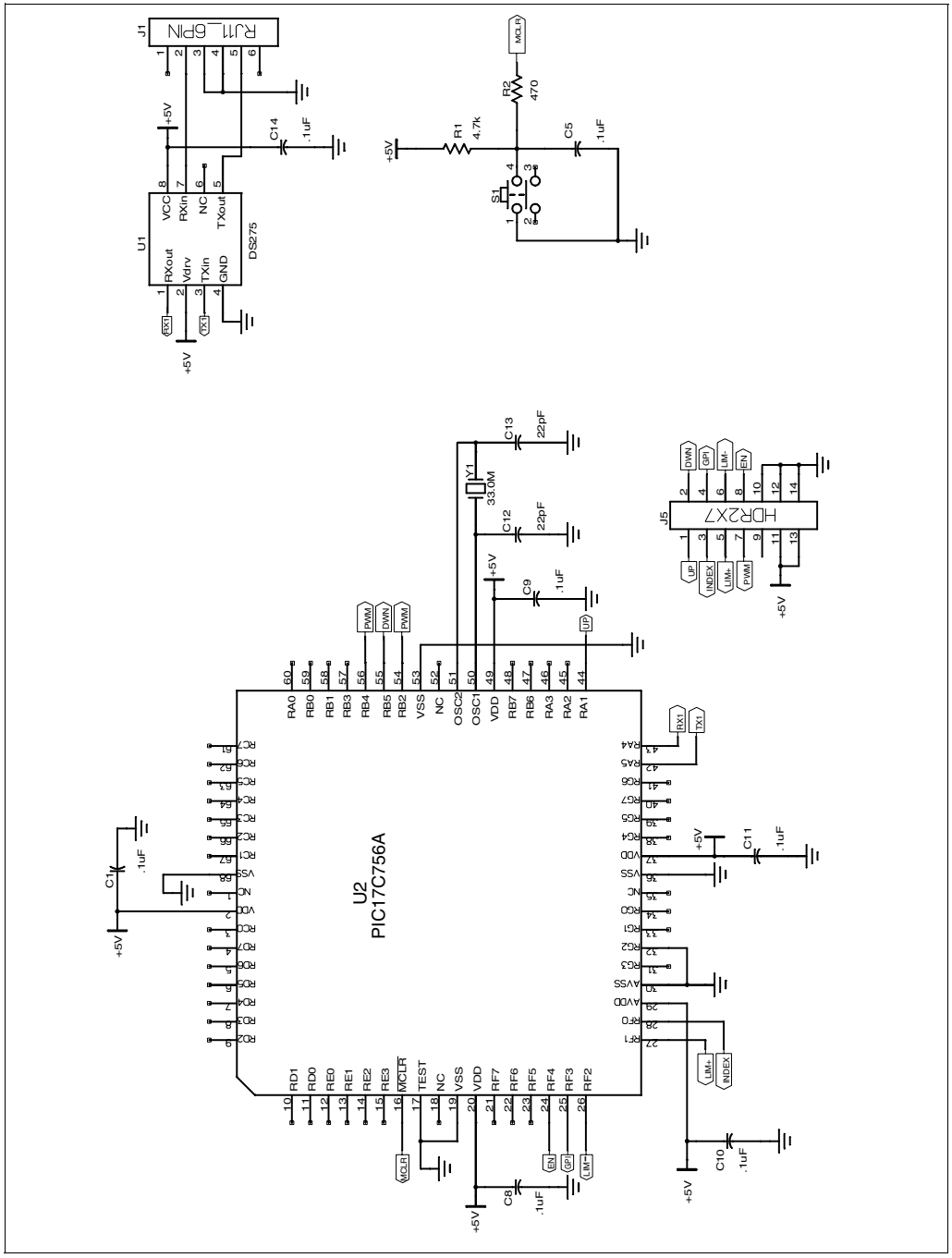


FIGURE A-2: SCHEMATIC 2

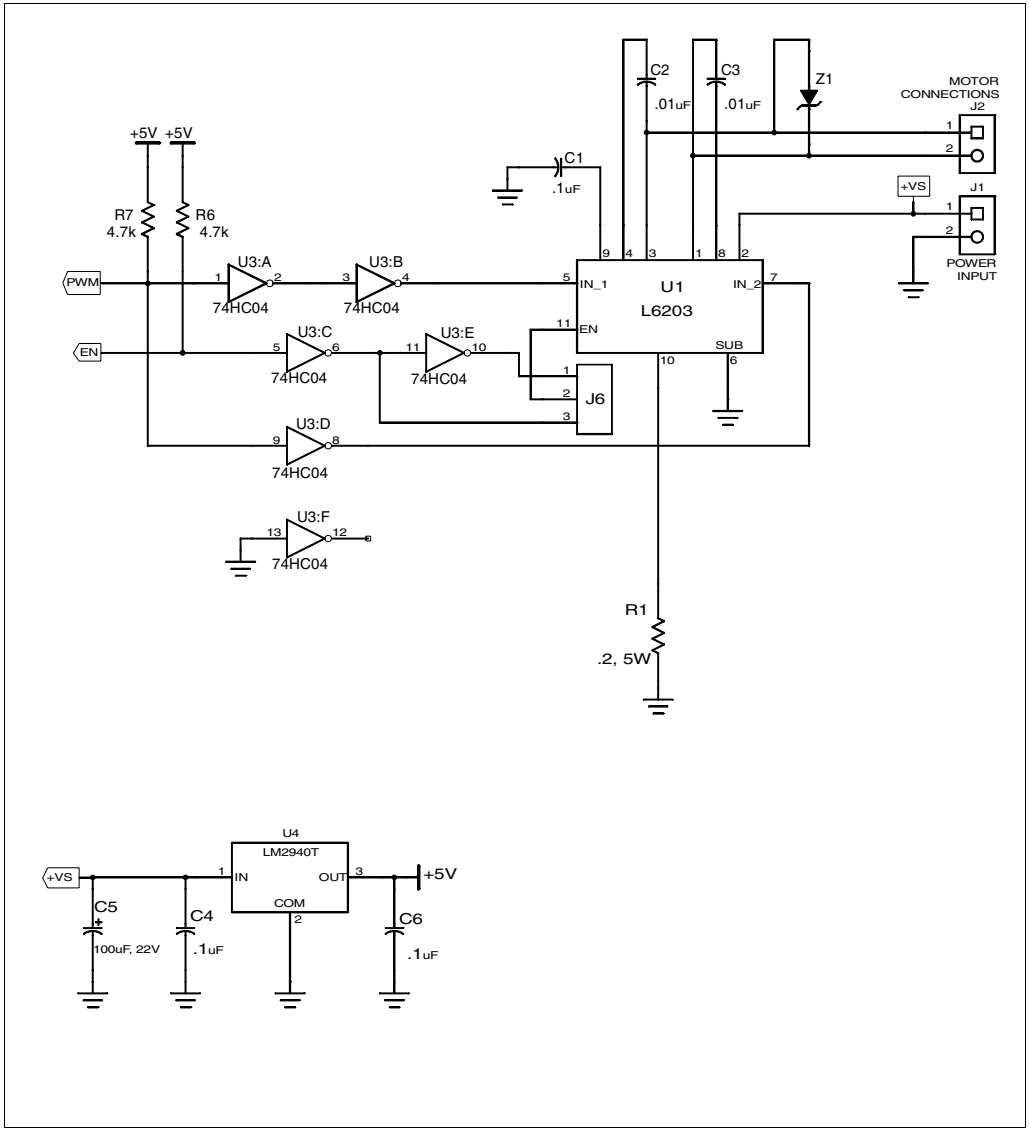
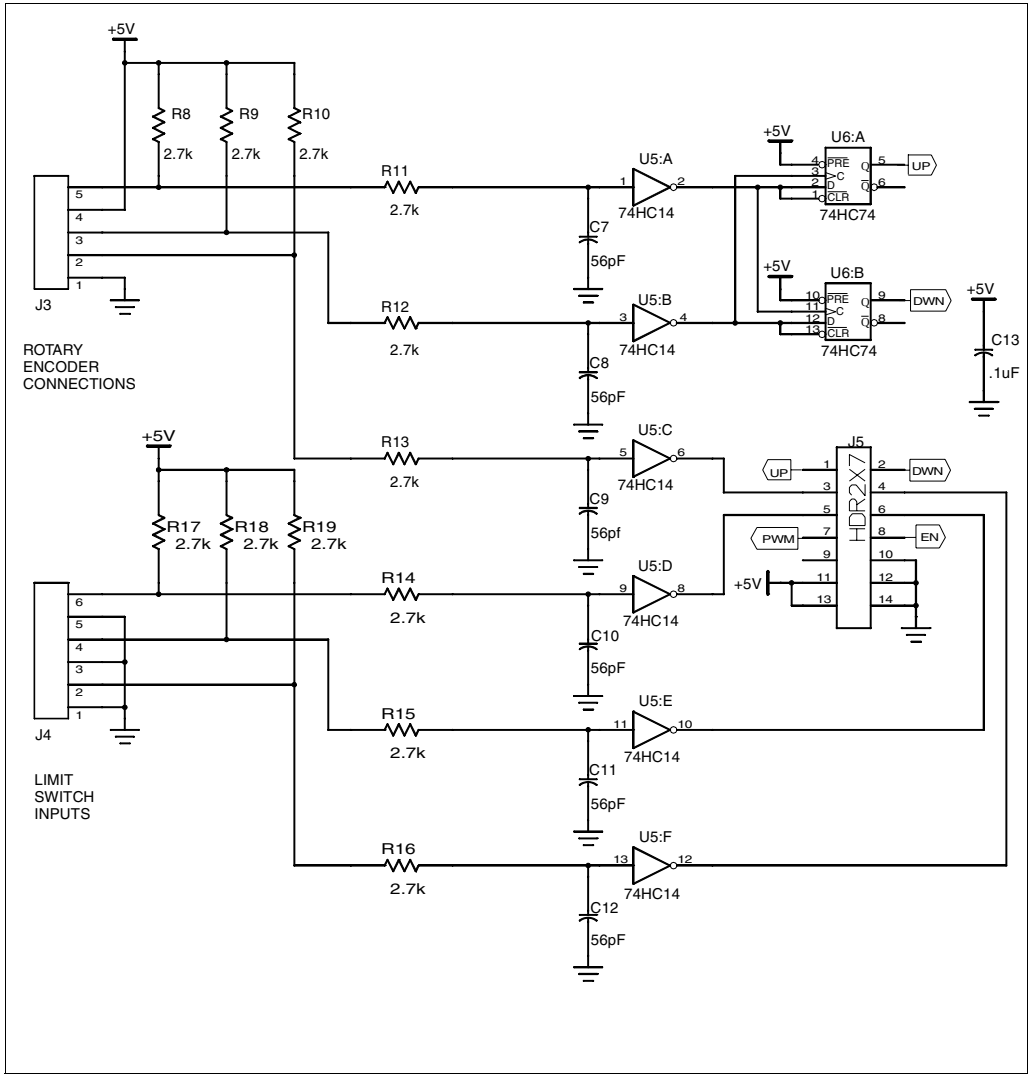


FIGURE A-3: SCHEMATIC 3



Please check Microchip's Worldwide website at www.microchip.com for the latest version of the source code.

APPENDIX B: SOURCE CODE

```
//-------------------------------------------------------------------
//              17motor.c
//              Written By:      Steve Bowling, Microchip Technology
//
//              This source code demonstrates the use of the PIC17C756A in a
//              brush-DC servomotor application and is written for the MPLAB-C17
//              compiler.  The following files should be included in the C17
//              project, which is compiled for the large memory model:
//
//              17motor.c        --
//              c0l17.o          --       startup code
//              idata17.o        --       initialized data support
//              p17c756.o        --       processor definition module
//              int756l.o        --       interrupt handler routines
//              pmc756l.lib      --       library functions
//              p17c756l.lkr     --       linker script
//
//-------------------------------------------------------------------

#include <p17c756.h>
#include <stdlib.h>
#include <usart16.h>
#include <string.h>
#include <timers16.h>
#include <captur16.h>
#include <pwm16.h>
#include <ctype.h>
#include <delays.h>
#include <mem.h>

#define  F  1
#define  W  0

const rom char start[] = "\r\n\r\n17C756A DC Servomotor";
const rom char ready[] = "\n\rREADY>";
const rom char error[] = "\n\rERROR!";

char inpbuf[8];                                 // input buffer for ASCII commands
char data[9];                                   // buffer for ASCII conversions
char command;                                   // holds the last parameter change
                                                // command that was received

unsigned char
i,                                              // index to ASCII buffer
udata,                                          // received character from USART
mode,                                           // determines servo mode
tempchar,
PRODHtemp,                                      // temp context saving for ISR
PRODLtemp,                                      // "
FSR0temp,                                       // "
FSR1temp;                                       // "

struct {                                        // holds status bits for servo
    unsigned            phase:1;                // first half/ second half of profile
    unsigned         neg_move:1;                // backwards relative move
    unsigned move_in_progress:1;                //
    unsigned   saturated:1;                     // servo output is saturated
    unsigned          bit4:1;
    unsigned              bit5:1;
    unsigned             bit6:1;
    unsigned             bit7:1;
} stat ;
```

```
int

tempint3,                                   //
tempint2,                                   //
tempint1,                                   //
tempint0,                                   //
UpCount,                                    // encoder up counts during sample period
DnCount,                                    // encoder down counts "      "      "
u0,u1,                                      // current and previous position error
kp,ki,kd,                                   // PID gain constants
integral,                                   // PID error accumulation
ypwm,                                       // duty cycle derived from PID calculation
multcnd,multplr,                            // holds values to be multiplied in mult()
velcom,vlim;                                // commanded velocity, velocity limit

unsigned int accel;                         // acceleration parameter for motion profile

union LONG
{
unsigned int ui[2];
int i[2];
char b[4];
};

union LONG
aarg,                                       // Used for math calculations.
barg,                                       //              "
ypid,                                       // Used to hold result of the PID
                                            // calculations.
position,                                   // Commanded position.
mposition,                                  // Actual measured position.
fposition,                                  // Final commanded position of motion
                                            // profile.
poserror,                                   // 32-bit position error calculated
                                            // in the PID
mvelocity,                                  // measured velocity
velact,                                     // current commanded velocity
phase1dist,                                 // total distance for first half of move.
flatcount;                                  // Holds the number of sample periods for
                                            // which the velocity limit was reached in
                                            // the first half of the move.


//  Function Declarations-----------------------------------------------

void main(void);                            // Required for the main function
void InitPorts(void);                       // Initializes ports/peripherals
void InitVars(void);                        // Initializes variable used in program
void DoCommand(void);                       // Parses input buffer after a <CR> was received
void ServoISR(void);                        // Performs the error calculations and PID
void UpdatePosition(void);                  // Updates the measured motor position
void UpdateTrajectory(void);                // Does the motion profile
void add32(void);                           // Performs a 32 bit addition
void sub32(void);                           // Performs a 32 bit subtraction
void mult(void);                            // Performs a 16 x 16 --> 32 multiplication
void ulitoa(unsigned int value1,            // Converts 32-bit value in two integers
unsigned int value0, char *string);         // to an ASCII string in hexadecimal
char ntoh(unsigned int value);              // format.


//---------------------------------------------------------------------


void main(void)
{
```

```
InitVars();
InitPorts();
Install_PIV(ServoISR);                          // Servo_ISR is installed as the
                                                // peripheral
Enable();                                       // int. handler.

putrsUSART1(start);
putrsUSART1(ready);

while(1)                                        // This is the main program loop
   {                                            // that polls USART1 for received
                                                // characters.
   if(PIR1bits.RC1IF)
      {
      switch(udata = ReadUSART1())
         {
         case 0x0d:  DoCommand();               // got a <CR>, so process the string
                     strset(inpbuf, 0);         // clear the input buffer
                     i = 0;                     // clear the input buffer index
                     putrsUSART1(ready);        // put a ready prompt on the screen
                     break;

         default:    inpbuf[i] = udata;         // put the received character in the
                     i++;                       // next buffer location and increment
                     if(i > 7)                  // the buffer index
                        {
                        putrsUSART1(ready);     // if we got more than 7 chars before a
                        strset(inpbuf, 0);      // <CR>, clear the input buffer and clear
                        i = 0;                  // the buffer index
                        }
                     else putcUSART1(udata);    // otherwise, echo the received character
                     break;                     //

         }                                      //end switch(udata)

      }                                         //end if(PIR1bits.RC1IF)


   }                                            //end while(1)
}                                               //end main

//---------------------------------------------------------------------


void DoCommand(void)                            // This routine parses the input buffer
{                                               // after a <CR> was received.
unsigned int num;


if(isdigit(inpbuf[0]) ||  inpbuf[0] == '-')     // Did we get a numerical input?
   {
   if(command)                                  // Was numerical input preceded
      {                                         // by a command to change a
      switch(command)                           // parameter?
         {
         case 'P':   kp = atoi(inpbuf);         // proportional gain change
                     break;

         case 'I':   ki = atoi(inpbuf);         // integral gain change
                     break;

         case 'D':   kd = atoi(inpbuf);         // differential gain change
                     break;

         case 'A':   accel = atoui(inpbuf);     // acceleration change
                     break;
```

```
        case 'V':   vlim = atoui(inpbuf);     // velocity limit change
                    break;

        case 'S':   PR2 = atoub(inpbuf);      // servo update timing change
                    break;

        default:    break;

        }
     command = 0;
     }

   else if(mode == 0) ypwm = atoi(inpbuf);     // manual mode:  write directly to PWM

   else if(mode == 1) velcom = atoi(inpbuf);   // velocity mode:  input data is velocity

   else if(mode == 2)                          // Input data is a relative movement
                                               // distance
      {                                        // distance for position mode.
      if(!stat.move_in_progress)               // Make sure no move is in progress.
         {
         phase1dist.i[1] = atoi(inpbuf);       // Load the 16-bit relative movement
                                               // distance into the upper
         phase1dist.i[0] = 0;                  // two bytes of phase1dist variable

         fposition.i[0] = position.i[0];       // Final position is commanded position
         fposition.i[1] = position.i[1]        // + relative move distance
                        + phase1dist.i[1];

         if(phase1dist.b[3] & 0x80)            // If the relative move is negative,
            {
            stat.neg_move = 1;                 // set flag to indicate neg. move

            _asm                               // and covert phase1dist to a positive
            comf     phase1dist+2,F            // value.
            comf     phase1dist+3,F
            clrf     WREG,F
            incf     phase1dist+2,F
            addwfc   phase1dist+3,F
            _endasm

            }
         else stat.neg_move = 0;               // Clear the flag for a positive move.

         _asm                                  // phase1dist now holds the total
            rlcf  phase1dist+3,W               // distance, so divide by 2
            rrcf  phase1dist+3,F
            rrcf  phase1dist+2,F
            rrcf  phase1dist+1,F
            rrcf  phase1dist+0,F
         _endasm

         flatcount.i[1] = 0;                   // Clear flatcount
         flatcount.i[0] = 0;

         stat.phase = 0;                       // Clear flag:  first half of move.
         stat.move_in_progress = 1;
         }
      }
   else;
   }

else switch(inpbuf[0])
   {
   case 'K':   if(inpbuf[1] == 'P') command = 'P';// If this is a parameter change,
```

```
            else                            // determine which parameter
            if(inpbuf[1] == 'I') command = 'I';
            else
            if(inpbuf[1] == 'D') command = 'D';
            else
            if(inpbuf[1] == 'A') command = 'A';
            else
            if(inpbuf[1] == 'V') command = 'V';
            else
            if(inpbuf[1] == 'S') command = 'S';
            break;

case 'W':   if(PORTFbits.RF4 == 0)
               {
               putrsUSART1("\r\nPWM ON");
               SetDCPWM1(512);
               }
            else
               {
               putrsUSART1("\r\nPWM OFF");
               }
            PORTF = PORTF ^ 0x10;           // enables or disables PWM amplifier
            break;

case 'R':   putrsUSART1("  Kp = ");        // Send all parameters to host.
            uitoa(kp, data);
            putsUSART1(data);

            putrsUSART1("  Ki = ");
            uitoa(ki, data);
            putsUSART1(data);

            putrsUSART1("  Kd = ");
            uitoa(kd, data);
            putsUSART1(data);

            putrsUSART1("  Vlim = ");
            uitoa(vlim, data);
            putsUSART1(data);

            putrsUSART1("  Acc. = ");
            uitoa(accel, data);
            putsUSART1(data);

            break;

case 'M':   putrsUSART1(" Manual Mode");    // Put the servomotor in manual mode.
            SetDCPWM1(512);
            mode = 0;
            break;

case 'V':   putrsUSART1(" Velocity Mode");  // Put the servomotor in velocity mode.
            velcom = 0;
            SetDCPWM1(512);
            position = mposition;
            fposition = position;
            mode = 1;
            break;

case 'P':   putrsUSART1(" Position Mode");  // Put the servomotor in position mode.
            SetDCPWM1(512);
            position = mposition;
            fposition = position;
            mode = 2;
            break;
```

```
   case 'L':   tempint0 = mposition.i[0];      // Send measured and commanded position
               tempint2 = position.i[0];       // to host.
               tempint1 = mposition.i[1];
               tempint3 = position.i[1];
               ulitoa(tempint1,tempint0,data);
               putrsUSART1(" Measured = ");
               putsUSART1(data);
               ulitoa(tempint3,tempint2,data);
               putrsUSART1("  Commanded = ");
               putsUSART1(data);
               break;

   case 'Z':   if(!stat.move_in_progress)      // Set measured position to 0.
                  {
                  if(mode) CloseTimer2();      // Disable interrupt generation.
                  position.i[1] = 0;
                  position.i[0] = 0;
                  mposition = position;
                  fposition = position;
                  WriteTimer0(0);
                  WriteTimer3(0);
                  mvelocity.i[1] = 0;
                  mvelocity.i[0] = 0;
                  UpCount = 0;
                  DnCount = 0;
                  if(mode) OpenTimer2(TIMER_INT_ON&T2_SOURCE_EXT);// Enable Timer2
                  }

               putrsUSART1(ready);
               break;

   default:    if(inpbuf[0] != '\0')
                  {
                  putrsUSART1(error);
                  }
               break;

   }

}

//---------------------------------------------------------------------


void ServoISR(void)
{
PRODHtemp = PRODH;                          // Save context for necessary registers
PRODLtemp = PRODL;
FSR0temp = FSR0;
FSR1temp = FSR1;

UpdatePosition();                           // Get new mposition, mvelocity values

if(mode)                                    // This portion of code not executed
       {                                    // in manual mode.
       UpdateTrajectory();                  // Do trajectory algorithm to get new
                                            // commanded position.
       aarg = position;                     // Subtract measured position
       barg = mposition;                    // from commanded position
       sub32();                             // to get 32 bit position error.

       poserror.b[2] = aarg.b[3];           // LSByte holds fractional encoder counts,
       poserror.b[1] = aarg.b[2];           // so shift everything right.
       poserror.b[0] = aarg.b[1];

       if (poserror.b[2] & 0x80)            // If position error is negative.
```

```
      {
      poserror.b[3] = 0xff;                // Sign-extend to 32 bits.

      if((poserror.i[1] != 0xffff) || !(poserror.b[1] & 0x80))
         {
         poserror.i[1] = 0xffff;           // Limit error to 16-bit signed integer
         poserror.i[0] = 0x8000;
         }
      else;
      }

   else                                    // If position error is positive.
      {
      poserror.b[3] = 0x00;

      if((poserror.i[1] != 0x0000) || (poserror.b[1] & 0x80))
         {
         poserror.i[1] = 0x0000;           // Limit error to 16-bit signed integer.
         poserror.i[0] = 0x7fff;
         }
   else;
      }

      u0 = poserror.i[0];                  // Put position error in u0.

      multcnd = u0;                        // Calculate proportional term
      multplr = kp;                        // of PID
      mult();
      ypid = aarg;

      if(!stat.saturated) integral +=u0;   // Bypass integration if saturated.

      multcnd = integral;                  // Calculate integral term of PID
      multplr = ki;
      mult();
      barg = ypid;
      add32();                             // Add integral term.
      ypid = aarg;

      multcnd = u0 - u1;                   // Calculate differential term of PID
      multplr = kd;
      mult();
      barg = ypid;                         // Add differential term
      add32();
      ypid = aarg;

      if(ypid.b[3] & 0x80)                 // If PID result is negative
         {
         if((ypid.b[3] < 0xff) || !(ypid.b[2] & 0x80))
            {
            ypid.i[1] = 0xff80;            // Limit result to 24-bit value
            ypid.i[0] = 0x0000;
            }
         else;
         }

      else                                 // If PID result is positive
         {
         if(ypid.b[3] || (ypid.b[2] > 0x7f))
            {
            ypid.i[1] = 0x007f;            // Limit result to 24-bit value
            ypid.i[0] = 0xffff;
            }
         else;
         }
```

```
          ypid.b[0] = ypid.b[1];              // Shift PID result right to get
          ypid.b[1] = ypid.b[2];              // upper 16 bits of 24-bit result in
          ypwm = ypid.i[0];                   // ypid.i[0]

          u1 = u0;                            // Save current error in u1
          }                                   // end if(mode)

stat.saturated = 0;                           // Clear saturation flag


if(ypwm > 500)
   {
   ypwm = 500;
   stat.saturated = 1;
   }
else if(ypwm < -500)
   {
   ypwm = -500;
   stat.saturated = 1;
   }

SetDCPWM1((unsigned int)(ypwm + 512));        // Write new duty cycle value

PRODH = PRODHtemp;                            // Restore context.
PRODL = PRODLtemp;
FSR0 = FSR0temp;
FSR1 = FSR1temp;

PIR1bits.TMR2IF = 0;                          // Clear flag that generated interrupt.
}

//-----------------------------------------------------------------------
// The relative distance travelled during the sample period is found using
// the following formula:
//
// mvelocity = (Timer0 - prev. Timer0) - (Timer3 - prev. Timer3)
//
// This is done so the timers do not have to be cleared each sample period
// and potentially cause counts to be lost.
//


void UpdatePosition(void)
{
mvelocity.i[0] = DnCount;                     // Add previous Timer3 value
mvelocity.i[0] -= UpCount;                    // Subtract previous Timer0 value

UpCount = ReadTimer0();                       // get new values from Timer0
DnCount = ReadTimer3();                       // and Timer3

mvelocity.i[0] += UpCount;                    // Add current Timer0 value
mvelocity.i[0] -= DnCount;                    // Subtract current Timer3 value

mvelocity.b[2] = mvelocity.b[1];              // Shift result left:  LSbyte is
mvelocity.b[1] = mvelocity.b[0];              // fractional
mvelocity.b[0] = 0;

if (mvelocity.b[2] & 0x80)                    // Sign-extend result
    mvelocity.b[3] = 0xff;
else
    mvelocity.b[3] = 0;

aarg = mposition;                             // Add velocity to measured position
barg = mvelocity;
add32();
mposition = aarg;
```

```
}

//-----------------------------------------------------------------------

void UpdateTrajectory(void)
{
if(mode == 1)                                   // If servomotor is in velocity mode.
   {
   if(!stat.saturated)                          // Don't update profile if saturated.
      {
      if(velact.i[1] < velcom)                  // If current velocity is less than
         {                                      // commanded velocity.
         aarg = velact;
         barg.i[0] = accel;                     // Accelerate
         barg.i[1] = 0;
         add32();
         velact = aarg;

         if(velact.i[1] > velcom)               // Don't exceed commanded velocity
         velact.i[1] = velcom;

         if(velact.i[1] > vlim)                 // Don't exceed velocity limit parameter
         velact.i[1] = vlim;
         }
      else
      if(velact.i[1] > velcom)                  // If current velocity exceeds commanded
         {                                      // velocity
         aarg = velact;
         barg.i[0] = accel;                     // Decelerate
         barg.i[1] = 0;
         sub32();
         velact = aarg;
         if(velact.i[1] < velcom)               // Don't exceed commanded velocity
         velact.i[1] = velcom;
         if(velact.i[1] < -vlim)                // Don't exceed velocity limit parameter
         velact.i[1] = -vlim;
         }
      else;

      aarg = position;                          // Add current commanded velocity to
      barg.i[0] = velact.i[1];                  // the commanded position
      if(velact.b[3] & 0x80)
      barg.i[1] = 0xffff;
      else barg.i[1] = 0;
      add32();
      position = aarg;
      }
   }

else if(mode == 2)
   {                                            // If we're in position mode.
   if(!stat.saturated)                          // Don't update profile if output is
                                                // saturated
      {
      if(!stat.phase)                           // If we're in the first half of the move.
         {
         if(velact.i[1] < vlim)                 // If we're still below the velocity limit
            {                                   // for the move
            aarg = velact;
            barg.i[0] = accel;
            barg.i[1] = 0;
            add32();
            velact = aarg;
            }
         else                                   // If we're at the velocity limit,
```

```
        {                                   // increment flatcount to keep track of
        _asm                                // time spent in flat portion of
        clrf     WREG,F                     // trajectory.
        incf     flatcount+0,F
        addwfc   flatcount+1,F
        addwfc   flatcount+2,F
        addwfc   flatcount+3,F
        _endasm
        }

      aarg = phase1dist;                    // go ahead and subtract the current
      barg.i[1] = 0;                        // velocity from the move distance to keep
      barg.i[0] = velact.i[1];              // track of the number of encoder counts
      sub32();                              // travelled during this sample period.
      phase1dist = aarg;

      aarg = position;                      // Add the current velocity to the
                                            // commanded position.
      if(stat.neg_move) sub32();
      else add32();
      position = aarg;

      if(phase1dist.b[3] & 0x80)            // If phase1dist has gone negative, the
      stat.phase = 1;                       // first half of the move has completed

      }

   else                                     // If we're in the second half of the
                                            // move.
      {
      if(flatcount.i[1] || flatcount.i[0])
        {
        _asm                                // If flatcount is not zero, decrement it.
        clrf     WREG,F
        decf     flatcount+0,F
        subwfb   flatcount+1,F
        subwfb   flatcount+2,F
        subwfb   flatcount+3,F
        _endasm
        }
      else
      if(velact.i[1])                       // If velact is not 0, decelerate.
        {
        aarg = velact;
        barg.i[0] = accel;
        barg.i[1] = 0;
        sub32();
        velact = aarg;
        }
      else                                  // flatcount is 0, velact is 0, so move is
        {                                   // over.  Set commanded position equal to
        position = fposition;               // the final position calculated at the
        stat.move_in_progress = 0;          // beginning of the move.
        }

      aarg = position;                      // Add current velocity to commanded
                                            // position.
      barg.i[1] = 0;
      barg.i[0] = velact.i[1];
      if(stat.neg_move) sub32();
      else add32();
      position = aarg;
      }
   }                                        // END if(!stat.saturated)
}                                           // END if(mode == 2)
```

```
else;

}

//---------------------------------------------------------------------

void add32(void)                                //
{
_asm

   MOVFP    barg+0,WREG
   ADDWF    aarg+0,F
   MOVFP    barg+1,WREG
   ADDWFC   aarg+1,F
   MOVFP    barg+2,WREG
   ADDWFC   aarg+2,F
   MOVFP    barg+3,WREG
   ADDWFC   aarg+3,F

_endasm
}

//---------------------------------------------------------------------

void sub32(void)                                //
{
_asm

   MOVFP    barg+0,WREG
   SUBWF    aarg+0,F
   MOVFP    barg+1,WREG
   SUBWFB   aarg+1,F
   MOVFP    barg+2,WREG
   SUBWFB   aarg+2,F
   MOVFP    barg+3,WREG
   SUBWFB   aarg+3,F

_endasm
}

//---------------------------------------------------------------------

void mult(void)                                 // Multiplies 16-bit values in multplr
{                                               // and multend.
_asm                                            // 32-bit result is stored in aarg

   movfp    multcnd+0,WREG
   mulwf    multplr+0
   movpf    PRODH,aarg+1
   movpf    PRODL,aarg+0

   movfp    multcnd+1,WREG
   mulwf    multplr+1
   movpf    PRODH,aarg+3
   movpf    PRODL,aarg+2

   movfp    multcnd+0,WREG
   mulwf    multplr+1

   movfp    PRODL,WREG
   addwf    aarg+1,F
   movfp    PRODH,WREG
   addwfc   aarg+2,F
   clrf     WREG,F
   addwfc   aarg+3,F
```

```
   movfp    multcnd+1,WREG
   mulwf    multplr+0

   movfp    PRODL,WREG
   addwf    aarg+1,F
   movfp    PRODH,WREG
   addwfc   aarg+2,F
   clrf     WREG,F
   addwfc   aarg+3,F

   btfss    multplr+1,7
   goto     $ + 5
   movfp    multcnd+0,WREG
   subwf    aarg+2,F
   movfp    multcnd+1,WREG
   subwfb   aarg+3,F

   btfss    multcnd+1,7
   goto     $ + 5
   movfp    multplr+0,WREG
   subwf    aarg+2,F
   movfp    multplr+1,WREG
   subwfb   aarg+3,F

   nop
_endasm
}

//-------------------------------------------------------------------------

void ulitoa(unsigned int value1, unsigned int value0, char *string)
{
unsigned int temp;                              // Converts 32-bit value stored in two
                                                // integers to an ASCII string in
temp = value1;                                  // hexidecimal format.
*string = ntoh(temp >> 12);
string++;

temp = value1 & 0x0f00;
*string = ntoh(temp >> 8);
string++;

temp = value1 & 0x00f0;
*string = ntoh(temp >> 4);
string++;

temp = value1 & 0x000f;
*string = ntoh(temp);
string++;

temp = value0;
*string = ntoh(temp >> 12);
string++;

temp = value0 & 0x0f00;
*string = ntoh(temp >> 8);
string++;

temp = value0 & 0x00f0;
*string = ntoh(temp >> 4);
string++;

temp = value0 & 0x000f;
*string = ntoh(temp);
string++;
```

```
*string = 0;

return;
}

//---------------------------------------------------------------------

char ntoh(unsigned int value)                   // Converts hexidecimal value to ASCII
{                                               // value.
char hexval;

if(value < 10) hexval = value + '0';
else if(value < 16) hexval = value - 10 + 'A';

return hexval;
}

//---------------------------------------------------------------------

void InitVars(void)
{
i = 0;

kp = 2000;
ki = 15;
kd = 6000;

vlim = 4096;
velcom = 0;
velact.i[1] = 0;
velact.i[0] = 0;
accel = 65535;

integral = 0;
mvelocity.i[1] = 0;
mvelocity.i[0] = 0;
UpCount = 0;
DnCount = 0;
position = mposition;
fposition = position;

stat.move_in_progress = 0;
stat.neg_move = 0;
stat.phase = 1;

mode = 0;
ypwm = 0;

strset(inpbuf,'\0');
}

//---------------------------------------------------------------------

void InitPorts(void)
{
ADCON1 = 0x0E;                                  // ensure port F is configured for
                                                // digital IO.
PORTF = 0x00;                                   // ensure port F is 0 before setting data
                                                // direction.
DDRF = 0x0f;                                    // RF<7:4> outputs, RF<3:0> inputs

PORTFbits.RF4 = 0;                              // ensure pwm amplifier is disabled!!!

// Up/Down Register Setup -----------------------

WriteTimer0(0);
```

```
WriteTimer3(0);
OpenTimer0(TIMER_INT_OFF&T0_EDGE_FALL&T0_SOURCE_EXT&T0_PS_1_1);
OpenTimer3(TIMER_INT_OFF&T3_SOURCE_EXT);
TCON2bits.CA1 = 1;

// PWM Setup -----------------------------------

OpenTimer1(TIMER_INT_OFF&T1_SOURCE_INT&T1_T2_8BIT);// set up timer1 for PWM timebase
OpenPWM1(0xff);                                   // start up PWM1
SetDCPWM1(512);                                   // set the initial PWM duty cycle
                                                  // to ~50%


PR2 = 0x08;                                       // Set Timer2 overflow period to 8
                                                  // for 3.9 kHz update at 33 MHz
OpenTimer2(TIMER_INT_ON&T2_SOURCE_EXT);           // Enable Timer2

// USART1 Setup --------------------------------

OpenUSART1(USART_TX_INT_OFF&USART_RX_INT_OFF&USART_ASYNCH_MODE&
           USART_EIGHT_BIT&USART_CONT_RX, 26); // open the serial port
                                                  // 19.2 kbaud @ 33 Mhz
}
```

NOTES:

System Design Considerations for Implementing a ROM Microcontroller

Author: *Rick Stoneking*
Microchip Technology Inc.

INTRODUCTION

When developing a system that will ultimately utilize a ROM-based microcontroller (MCU), it is still typical to make use of an EPROM-based MCU during the final stages of the design. Initial development may also include the use of some type of emulator system, but prototype units normally make use of a windowed EPROM or OTP EPROM MCU, and the design is optimized/validated based upon the performance of the EPROM-based device without, in many cases, taking into consideration potential differences in the performance of the ROM-based device that will ultimately be used.

CAUSE OF OTP VS. ROM DIFFERENCES

While MCU manufacturers go to great lengths to ensure that the performance differences of EPROM vs. ROM devices are minimized, there are external factors that historically have prevented fully achieving this goal. There are a number of key factors that can contribute to differing performance between the two types of devices, which include:

- **Operating Voltage Range:**

  ROM devices operate to a lower V_{DDMIN} due to the difference in physics between EPROM and ROM memory cells.

- **Parametrics:**

  ROM and EPROM devices are not manufactured using the same fabrication process, leading to subtle differences in parametric performance.

- **Functional Operation:**

  One device may have design changes implemented to improve performance or correct errata that exists on the other device.

Each of these issues is discussed in more detail in the appropriate sections that follow.

Designers who are developing systems using EPROM products that are targeted to move to ROM devices as production volumes increase, or who find themselves needing to convert an existing EPROM-based design to ROM, should thoroughly review this application note to determine if the potential for problems exist. This document is not intended to be an all encompassing list of all possible issues, it is simply a reference resource for key items that have previously been identified as potentially causing problems.

OPERATING VOLTAGE RANGE

EPROM devices operate at V_{DD} levels above ~2.3V limited by the device physics of an EPROM cell. The ROM devices do not have this limitation and, therefore, typically operate down below 2.0V. When designing a low voltage system and developing/validating the design using an EPROM device, it is necessary to use a higher V_{DD} level than that which will actually be used in the final design. The gain of the internal transistors are sensitive to the V_{DD} value and this can lead to functional performance differences in the oscillator start-up/stabilization time, the watchdog timer speed. V_{IH}/V_{IL}, and V_{OH}/V_{OL} levels. Each of these issues is discussed in greater detail under the 'Parametrics' section.

The system designer(s) should ensure adequate margin to the published specifications when using EPROM-based devices for development, and the use of ROM prototypes is highly recommended for low voltage application validation.

PARAMETRICS

The parametric performance of the ROM equivalent of an EPROM-based device may vary due to the processes used to fabricate the two different devices. There are a number of different scenarios that lead to the two devices being fabricated using different process technologies. First, ROM devices do not require several of the process steps required to make an EPROM device, so the processes are different by definition. Second, ROM devices are often manufactured using different starting wafer sizes and/or different process geometries. These options help maximize the cost savings that can be realized with ROM devices.

All of these may lead to some amount of variation in the parametric performance between the EPROM and ROM devices. The manufacturer ensures that both the ROM and EPROM devices meet the datasheet specifications so that drop in compatibility is maintained. However, it is sometimes the case that a design

becomes dependent upon the actual parametric performance of a device instead of being designed to operate under the worst case specifications. This can lead to problems when developing a ROM application using an EPROM, or if trying to port a EPROM product to ROM to realize a cost reduction.

OSCILLATOR PERFORMANCE

Oscillator performance is a key parameter that may vary relatively significantly between the EPROM and ROM devices. The operation of the oscillator is highly dependent upon the internal transistor gains, which are determined by the process technology used during fabrication.

The transistor gains of the oscillator circuit effect oscillator start-up time and the oscillator stability with a given set of external components (crystal/resonator, capacitors, resistors). It is absolutely critical that the system designer(s) make every effort to verify the performance of the ROM device with the intended crystal/resonator design. This is highly recommended for oscillator verification whenever possible.

Another potential issue is, if the VDD ramp rate is relatively slow, the oscillator start-up timer may start sooner, relative to the start of the VDD ramp.

WATCHDOG TIMER (WDT)

The watchdog timer (WDT) is another function which can be highly sensitive to the parametrics of the process used to fabricate the device. The WDT utilizes an internal free running RC oscillator. The values of the internal resistor and capacitor may vary relatively significantly between the EPROM and ROM devices. It is, therefore, necessary to either allow for this in the selection of the WDT time-out value, or verify the design using actual ROM devices, if possible.

CURRENT CONSUMPTION

The current consumption between EPROM and ROM devices may also vary as a result of parametric differences in the processes. This includes both IDD and IPD values. Again, the manufacturer ensures that both devices meet the datasheet specifications, but designs that are very power sensitive should be evaluated using actual ROM devices, if possible to verify that the final design meets the current and power targets.

VOLTAGE THRESHOLDS

Another area where process parametrics may cause subtle differences in device operation is related to the VIL/VIH and VOL/VOH values of the device. Because these levels are a dependent upon the internal transistor thresholds, which is a function of the process used to manufacture the device, careful consideration should be given to the input and output level specifications, and the system should be designed to work with the specified worst case values.

ELECTROSTATIC DISCHARGE (ESD) PERFORMANCE

In some cases, there may be a difference in the actual ESD performance of the ROM versus EPROM devices. This may lead to problems in some designs, where ESD events are likely or common. The system designer should check the ROM device datasheet to determine if there is a difference in the ESD specification and, for applications that are expected to be particularly susceptible to ESD, should perform system validation with ROM devices, if possible.

FUNCTIONAL OPERATION

Functional operation differences between EPROM and ROM devices that are meant to be equivalent occasionally do occur. These differences are typically due to the fact that one of the devices (usually the EPROM) is developed and released first and contains some errata concerning actual functional performance. The second device typically implements fixes for some or all the known errata and, therefore, does not function identically to the other.

In other cases, changes or improvements may have been implemented to enhance a device but the enhancements may not have been released to production on both devices, so there is some period where the devices do not function identically.

It should also be noted that it should not be assumed that any or all errata for the EPROM device has been, or will be, corrected in the ROM device, and it is also possible that new errata is introduced on the ROM device that did not exist on the EPROM device.

Functional differences are often related to the operation of one of the peripheral blocks including:

- USART
- SSP
- PWM
- Timers
- $\overline{\text{MCLR}}$ operation
- A/D Converter

In all cases, the system designer(s) should specifically request any errata that exists for each of the two devices, as well as any known device specific issues between the EPROM and ROM versions of the device being used. And finally, ROM prototypes should be used whenever possible for final system validation.

ROM PROTYPES

Microchip offers customers a ROM prototype service, which allows systems in the latest stages of design validations to be checked out using a ROM PIC rather than an EPROM-based micro. This should be used if there are any concerns about the functional or parameter differences between the EPROM micro and the intended ROM device.

SUMMARY

When developing a new ROM application using an EPROM-based MCU, or when attempting to move an established EPROM-based design to ROM to reduce costs, there are a number of key factors to be considered to minimize problems and ensure a reliable ROM design. The ideas presented in this application note are not intended to be all inclusive, but do represent key issues that have been identified in the past as presenting potential problems. It can not be stressed enough that actual ROM devices should be used for system/design validation whenever possible. This alone significantly reduces the risk of unanticipated application performance issues occurring in the future. It is also key that all hardware be designed so that acceptable operation at worst case device specifications is ensured.

NOTES:

AN724

Using PICmicro® MCUs to Connect to Internet via PPP

SOFTWARE LICENSE AGREEMENT

The software supplied herewith by Microchip Technology Incorporated (the "Company") for its PICmicro® Microcontroller is intended and supplied to you, the Company's customer, for use solely and exclusively on Microchip PICmicro Microcontroller products.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

Author: *Myron Loewen*
for Microchip Technology Inc.

INTRODUCTION

PICmicro microcontrollers (MCU) are suitable for low-cost connections to the Internet. The desire to connect everything to the Internet focuses the price reduction challenge on the Internet interface. Typically, the interface is an overpowered embedded PC running a bulky operating system and memory intensive applications. For low-cost applications that handle smaller amounts of data, a much better choice is the Microchip family of PICmicro MCUs. This application note will show how these little processors are capable of connecting to the Internet with resources to spare for controlling the original application.

The software will dial into the Internet and try to connect to the server using Point to Point Protocol (PPP). It continues pinging once every 30 seconds to keep the connection open while responding to ping requests. With the Internet Protocol (IP) connection established you can add your own algorithms for traceroute, Trivial File Transfer Protocol (TFTP), Simple Network Management Protocol (SNMP), or get the time and date accurate to a millisecond.

Since there are lots of good books and free Internet resources to describe how the Internet works, this application note will focus on the less publicized details of negotiating PPP. Another common protocol used to connect to the Internet by modem is the Serial Line Internet Protocol (SLIP). PPP was chosen for this application note instead of the simpler SLIP because it is much more versatile. PPP has the advantage of not requiring a unique login script for most servers. Another advantage of PPP is line quality monitoring. Although not implemented in this algorithm, it is useful when reliable communications is a top priority. The most important reason is to maintain compatibility with Internet Service Providers by riding the popularity of desktop operating systems, which use PPP by default.

This Internet interface requires a physical connection to a local Internet Service Provider (ISP) with a serial line or modem. The rest is all software. The algorithm requires about 145 bytes of RAM and 2170 words of ROM. The amount of processor time available for other tasks will depend on the processor's clock speed and baud rate of the serial connection. The algorithm takes time for each received or transmitted character and extra time to process or create a data or control packet.

This algorithm does not include Transmission Control Protocol (TCP) which is required for email, Telnet, web browsing, and File Transfer Protocol (FTP). These algorithms could be added but they require a processor with a lot more RAM and ROM. This algorithm will not connect to every server; it attempts an unscripted PPP login and falls back to a generic script. If it fails, some login tweaking or implementing more Link Control Protocol (LCP) options will usually bring up the connection.

FIGURE 1: PHOTO OF PROTOTYPE CONNECTED TO THE INTERNET

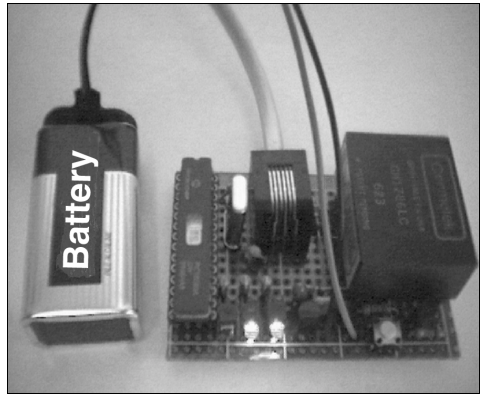
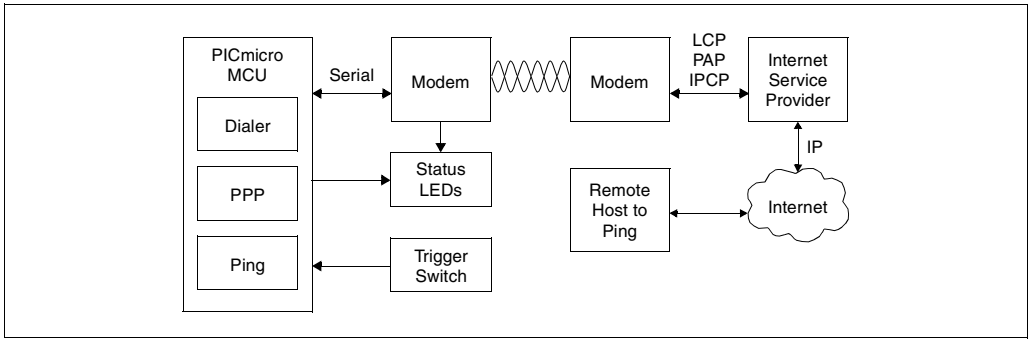


FIGURE 2: BLOCK DIAGRAM



INTERNET PROTOCOLS

The Internet is just a very large bunch of many types of computers connected in a variety of ways. What makes it all work is the thousands of standards and conventions that all computers follow. Most standards are documented and freely available on the Internet. Table 5 lists the standards needed for this algorithm and where to find them.

Data gets around the Internet in specially marked packets that are passed from computer to computer. The packets indicate the type of data they contain such as a part of a web page or email. Each packet gets stuffed in its own envelope specially marked to get it to the right program on the remote computer. This is important because you may be running several web browsing windows simultaneously on one machine. The type of data determines if the envelope is the simpler UDP type or a more robust TCP type. TCP packets generate extra packets to open and close the connection and resend lost packets.

Each computer gets a unique Internet address that looks like 10.241.45.12, and works much like a postal mailing address. The envelope is stuffed in a larger envelope with the source and destination addresses written on the front. This is like international mail, the address will get it anywhere on the Internet.

But the Internet works more like passing notes in class. The larger envelope goes into a bigger envelope with your friend's name on it. Your friend opens the envelope and checks the Internet address. If he is the recipient, he processes it, otherwise he puts it in a new big envelope. From the Internet address he can tell which direction to send the envelope and puts the name of his neighbor, in that direction, on the front. The process repeats until the envelope makes it to the correct Internet address across the class, or gets lost along the way.

In this algorithm only the ping packets travel this way. The other packets have the same format but are just exchanged locally between this algorithm and the Internet server it dials up. These packets are discussed in the next few sections and are used by both ends to configure the serial link options.

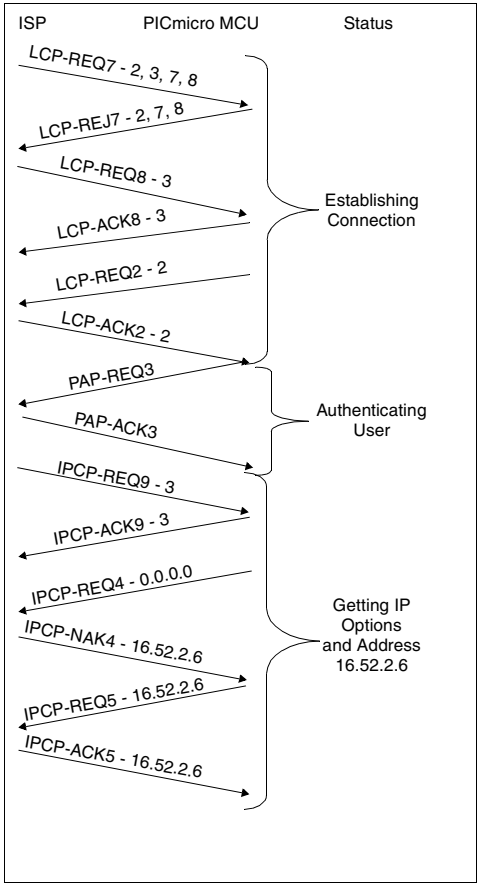
PPP requires the serial data format to be set to eight data bits with no parity. The PPP data is sent as packets that start and stop with the tilde character (~) or in hexadecimal 0x7E. Because ~ has a special meaning, any other instance of ~ is replaced by the }^ escape sequence. The escape sequence works by transmitting two characters instead of the original, first the } and then the original exored with the space character, or in hexadecimal 0x20. Because the } has a special meaning, it too is also escaped in the same way resulting in the 2-character sequence }]. For compatibility with all serial links, the control characters 0x00 to 0x1F can also be optionally mapped into the 2-character escape sequence. For more complete details, read RFC 1662 *PPP in HDLC-like Framing*.

The PPP connection can be broken into several phases. First, if the link is dead, carrier detect from the modem is one of the stimuli that starts the link establishment phase. This phase uses Link Control Protocol (LCP) to detect and negotiate link options with the remote computer.

Next the authentication phase verifies the User ID and password with Password Authentication Protocol (PAP). Although not one of the phases, this is where ISPs negotiate compression protocols. The final phase is the network layer protocol. Each protocol such as IP, is configured with its control protocol like IPCP.

Control protocols are very similar for LCP, PAP, CCP, and IPCP but the protocol field is different and the options have different meanings. Each packet can request, deny, or accept a list of options. Negotiation starts with either side requesting a list of options in a request (REQ) packet. Each option consists of an option type byte, length byte, and option parameters. If the receiving end likes all the options, it replies with an acknowledge (ACK) packet.

FIGURE 3: PPP NEGOTIATION WITH REQ, ACK, NAK, AND REJ PACKETS



If it doesn't like some parameters, it responds with a not acknowledge (NAK) packet that repeats all the options it rejects and replaces the rejected parameters with acceptable values. If required options are missing those are added to the NAK reject list.

If some options are not recognized or are considered non-negotiable they are rejected with the REJ packet that lists all the bad options. The first side updates its option list and retransmits requests until it gets an ACK reply packet. The other side can start negotiations at any time and the resulting link may have different options for each direction. The terminate, code reject, protocol reject, echo, and discard control packet types are not implemented in this algorithm. The details are broken down into a section for each control protocol.

TABLE 1: ACRONYMS

Acronym	Description
ACK	Acknowledgement
CCP	Compression Control Protocol
CRC	Cyclic Redundancy Check
CHAP	Challenge-Handshake Authentication Protocol
DNS	Domain Name System
DTR	Data Terminal Ready
FTP	File Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IPCP	Internet Protocol Control Protocol
ISP	Internet Service Provider
LCP	Link Control Protocol
MRU	Maximum Receive Unit
NAK	Negative Acknowledgement
PAP	Password Authentication Protocol
PPP	Point-to-Point Protocol
REJ	Reject
REQ	Request
RFC	Request For Comment
SLIP	Serial Line Internet Protocol
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
TTL	Time To Live
UDP	User Datagram Protocol

DIAL-UP SCRIPT

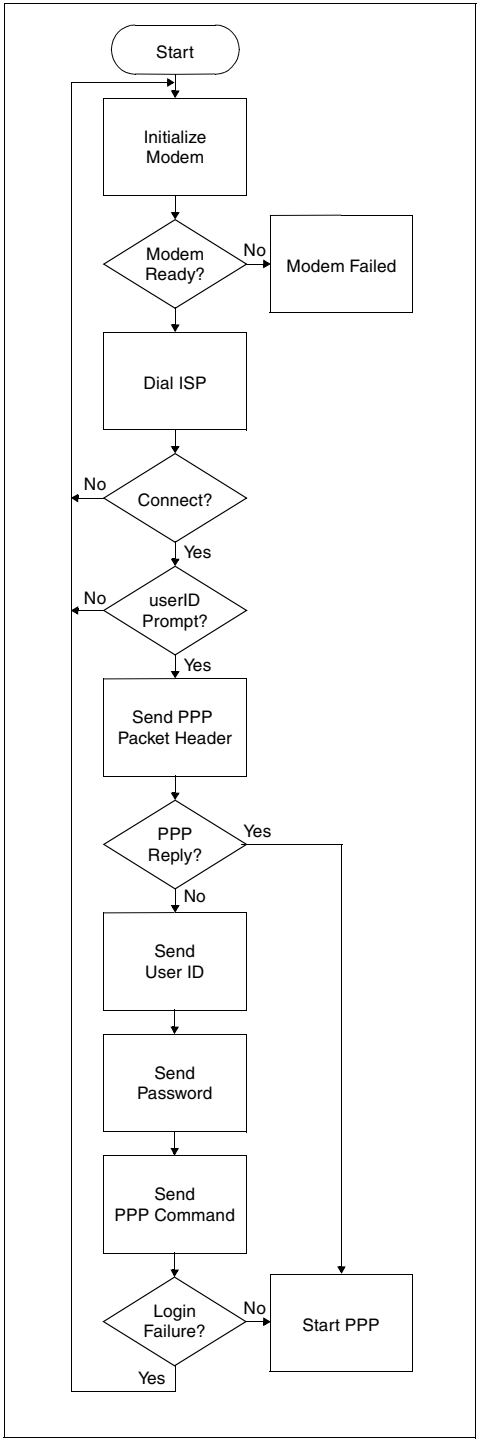
We cannot start link negotiations without a physical connection to the Internet. In this algorithm, a dial-up modem makes the connection and then the link is negotiated. The modem dial script could be removed for circuits with a direct connection to the server's serial port. The `sendwait (command, test, timeout)` routine does most of the work for the script. The script sends the command string and then returns when it receives the test string or the time expires. The timeout units are 10 milliseconds, thus a timeout of 100 is 1000 milliseconds or 1 second.

First the modem is taken into command mode with a pause, then three plus characters (+) and another pause. The | character in the command string causes a 1-second pause; use them where required in any `sendwait()` command string. Then, the `ATH` command hangs up the modem in case it was already off hook. The `ATZ` and `AT&F` commands then reset the modem and restore factory default settings. `ATS11=50` speeds up the tone dialing to reduce connect time. The modem has 3 seconds to echo the command before the algorithm aborts and assumes the modem is not functioning.

The algorithm indicates dialing by flashing the status LED. Then it sends the dial string to the modem and waits 30 seconds for the modem to respond with the connect message before the algorithm gives up and tries the whole process again. If the dial string contains pauses, or the modem is faster than 2400 baud you may need to increase the timeout. When the modem connects, the connection LED stays on steady, otherwise it is turned off.

The script waits 10 seconds for a colon followed by a space to detect when the server is prompting for the userID. Whether it gets the prompt or not, it sends the first PPP packet. This makes use of the fact that most servers switch to PPP login instead of further text prompts when they get the first few characters of a PPP packet instead of a valid User ID. If the bait is taken, the script ends and PPP negotiation begins. Otherwise the script continues by sending the User ID, password, and command to enter PPP. The IP address is not captured at this time because the IPCP negotiations will capture it later.

FIGURE 4: PPP LOGIN FLOW CHART



LCP OPTIONS

The LCP options are negotiated first to establish the link. A sample packet is shown in Figure 5. It has the normal PPP header of 0x7E 0xFF 0x03 followed by 0xC0 0x21 to indicate that the protocol is LCP. The LCP packet consists of a code, identification, length, and a list of options to configure followed by the standard 2 byte PPP CRC. The code is a byte to indicate the meaning of the packet. A list of codes is found in Table 2. The identification byte is incremented after each negotiation request, which makes requests unique and connects them to the correct reply. The 16-bit length is the number of bytes in the LCP packet, four for the header plus the sum of the lengths of each option.

The list of possible options is found in Table 3. Each option is sent as a one byte option type, followed by a one byte option length, and an optional parameter. The option length is two for the option header plus the number of bytes in the parameter. Here is a brief description of the more common options:

1 **Maximum-Receive-Unit** – The 2 byte parameter is the maximum size of PPP packets. It would be nice to make this value very small to conserve buffer space in the limited PICmicro MCU RAM. However, the minimum allowable value of 576 is much too big to help. Since the MRU option has no benefit and can be safely left at the 1500 default, this algorithm doesn't waste code space to support it. Note that packets longer than the 47 byte buffer size are truncated to fit, and typically the longest packet to handle is about 40 characters. Some ping packets are much longer but they are quite tolerant of losing the extra padding characters.

2 **Async-Control-Character-Map** – The 4-byte parameter in this option adds up to 32 bits: each bit represents one of the ASCII control characters from 0 to 31. Starting with the most significant bit as character 31 and the least significant as character 0. If the bit for the character is a one then that character must be transmitted as a } sequence. This way the server and client can decide to escape only the characters which may cause problems instead of wasting bandwidth escaping all control characters. Even characters that do not need to be escaped may be, this algorithm exploits that to simplify the software and to transmit all control characters as two bytes.

3 **Authentication-Protocol** – This option chooses the method of sending the password. Unless you have already logged in with the script, this option will be required. A parameter value of 0xC023 selects Password Authentication Protocol (PAP) which sends a packet containing the User ID and password in plain text. A parameter value of 0xC223 selects the Challenge Handshake Authentication Protocol (CHAP) in which the User ID is sent in plain text and the password encrypts and returns a random string from the server. On the server, the password encrypts the same string; if the two results match, the user is logged in. For simplicity, this algorithm only supports the PAP method. So far no ISP has forced it to use the CHAP method.

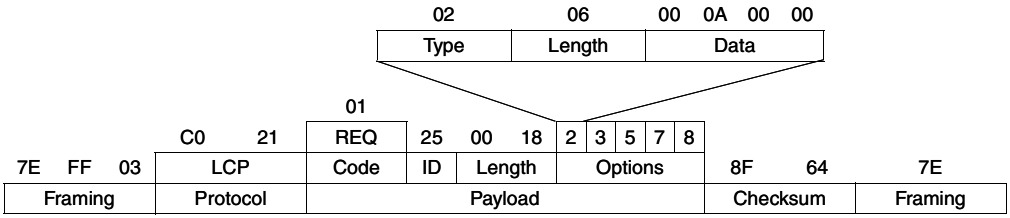
5 **Magic-Number** – This option did not need to be implemented for the PPP negotiations to converge. The 4 byte parameter is a random number; if identical to the server's, then both ends choose another random number. Assuming good randomness, the chance of random numbers not being unique after three iterations is so low that the transmission path is assumed to be looped back, just echoing packets sent.

7 **Protocol-Field-Compression** – This option has no parameters. If requested, the acknowledging side may transmit future PPP packets with the first byte of the 2-byte protocol field left out. This is meant to save bandwidth. It is easy to uncompress - if an odd byte arrives at the start of the protocol field it must be the second byte since the first byte is always even, and the second is always odd. A zero is inserted for the missing first character.

8 **Address-and-Control-Field-Compression** – This option has no parameters. However, if requested, the acknowledging side may transmit future PPP packets with the second and third bytes, 0xFF and 0x03, left out. This is also meant to save bandwidth. It is also easy to decompress because if the first character in the packet is not 0xFF, a 0xFF is inserted first, if the second character is not 0x03, a 0x03 is inserted first.

The other options didn't need to be implemented to make the Internet connection, but as standards evolve in the future a missing option could prevent login. Note that only options up to number 16 can be negotiated without modifying the `TestOptions()` routine. It has a one word parameter called option in which each of the 16 bits indicate acceptance of an option. For example, if the Most Significant Bit (MSB) is set, then option 16 is accepted; if bit 0 is cleared, then option 1 is rejected. LCP is complete when both sides of the connection have their list acknowledged by the other side.

FIGURE 5: A SAMPLE LCP REQUEST PACKET



The complete packet with more option details:

```
0000:  7E  FF  03  C0  21  01  25  00  18  02  06  00  0A  00  00  03
0010:  04  C0  23  05  06  71  AE  BE  D2  07  02  08  02  8F  64  7E
```

Same LCP packet with } escape sequences:

```
0000:  7E  FF  7D  23  C0  21  7D  21  25  7D  20  7D  38  7D  22  7D
0010:  26  7D  20  7D  2A  7D  2A  7D  20  7D  23  7D  24  C0  23  7D
0020:  25  7D  26  71  AE  BE  D2  7D  27  7D  22  7D  28  7D  22  8F
0030:  64  7E
```

TABLE 2: LCP NEGOTIATION CODES

Type	Packet Type	Details
0	Vendor Specific	RFC2153
1	Configure-Request	RFC1661
2	Configure-Ack	RFC1661
3	Configure-Nak	RFC1661
4	Configure-Reject	RFC1661
5	Terminate-Request	RFC1661
6	Terminate-Ack	RFC1661
7	Code-Reject	RFC1661
8	Protocol-Reject	RFC1661
9	Echo-Request	RFC1661
10	Echo-Reply	RFC1661
11	Discard-Request	RFC1661
12	Identification	RFC1570
13	Time-Remaining	RFC1570

TABLE 3: LCP OPTIONS

Type	Configuration Option	Details
0	Vendor Specific	RFC2153
1	Maximum-Receive-Unit	RFC1661
2	Async-Control-Character-Map	RFC1662
3	Authentication-Protocol	RFC1661
4	Quality-Protocol	RFC1661
5	Magic-Number	RFC1661
6	Quality-Protocol	Deprecated
7	Protocol-Field-Compression	RFC1661
8	Address-and-Control-Field-Compression	RFC1661
9	FCS-Alternatives	RFC1570
10	Self-Describing-Pad	RFC1570
11	Numbered-Mode	RFC1663
12	Multi-Link-Procedure	Deprecated
13	Callback	RFC1570
14	Connect-Time	Deprecated
15	Compound-Frames	Deprecated
16	Nominal-Data-Encapsulation	Deprecated
17	Multilink-MRRU	RFC1990
18	Multilink-Short-Sequence-Number-Header	RFC1990
19	Multilink-Endpoint-Discriminator	RFC1990
20	Proprietary	
21	DCE-Identifier	RFC1976
22	Multi-Link-Plus-Procedure	RFC1934
23	Link Discriminator for BACP	RFC2125
24	LCP-Authentication-Option	
25	Consistent Overhead Byte Stuffing (COBS)	
26	Prefix elision	
27	Multilink header format	
28	Internationalization	RFC2484
29	Simple Data Link on SONET/SDH	

PAP OPTIONS

The PAP details can be found in RFC 1334. For this algorithm they were simplified to one packet exchange. The PAP packet is similar to LCP with 0xC023 instead of 0xC021 in the protocol field. Instead of negotiating options, only the User ID and password are sent as a request. If the server acknowledges, then the user is logged in. A NAK reply would mean that the User ID or password is incorrect. The format can be seen in Figure 6. The first payload byte is the length of the User ID, followed by the User ID. The password is appended in the same way: Length first followed by password.

FIGURE 6: A SAMPLE PAP REQUEST PACKET

			C0 23	01	04	0014	06 userid	08 password			
7E FF 03			PAP	Code	ID	Length	User ID	Password	58	3D	7E
Framing			Protocol	Payload					Checksum		Framing

The complete packet:

```
0000: 7E FF 03 C0 23 01 04 00 14 06 75 73 65 72 69 64
0010: 08 70 61 73 73 77 6F 72 64 58 3D 7E
```

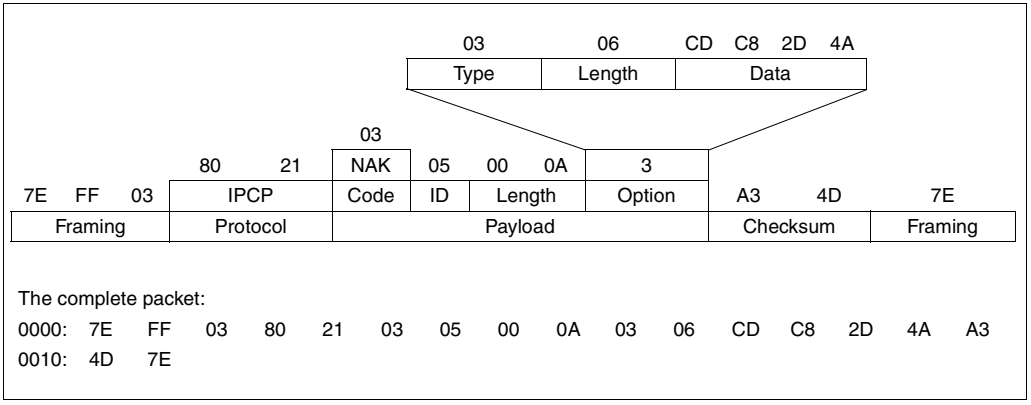
IPCP OPTIONS

After LCP is negotiated and PAP is accepted, the Internet Protocol must be configured. The options are for IP address and IP Compression with more details in RFC 1332. IP address is option three and its 4-byte parameter is the Internet address of this node. The server typically sends a request with option three followed by the IP address. Otherwise, the address is found by requesting an invalid choice like 0.0.0.0 and the server replies with a NAK and option three with the correct address. A sample packet is shown in Figure 7. Some server implementations may request IP Compression Protocol option type two. These requests are rejected because TCP is not implemented in this algorithm. Table 4 shows the IPCP configuration option types.

TABLE 4: IPCP CONFIGURATION OPTION TYPES

Type	Configuration Option	Details
1	IP-addresses	Deprecated
2	IP-Compression-Protocol	RFC1332
3	IP-address	RFC1332
4	Mobile-IPv4	RFC2290
129	Primary DNS Server address	RFC1877
130	Primary NBNS Server address	RFC1877
131	Secondary DNS Server address	RFC1877
132	Secondary NBNS Server address	RFC1877

FIGURE 7: A SAMPLE IPCP NAK PACKET



The complete packet:

0000: 7E FF 03 80 21 03 05 00 0A 03 06 CD C8 2D 4A A3

0010: 4D 7E

CCP OPTIONS

Some servers will try to negotiate compression, but since this algorithm is optimized for size instead of speed, these requests are rejected. The compression algorithms are complex and in some cases proprietary, yet have little benefit on the short packets used in this algorithm. Choosing the puddle jumper option type 3 means that no compression or decompression is required.

ICMP COMMUNICATIONS

The Internet Control Message Protocol messages are sent with full IP packets, an example is shown in Figure 8. This protocol is used to implement ping, but it has many other uses that you can read about in RFCs 792 and 950. Ping works by sending a packet to a remote Internet address and waiting for the reply, like radar or sonar from which it gets the name. This algorithm pings a fixed address once every 30 seconds to maintain an ISP connection. During this time it also responds to pings initiated by remote devices on the Internet.

The description of this packet is better understood as two parts, the IP header and the ICMP message. The packets discussed up to this point were just for setting up the serial link and never made it past the server. A lot more information is packed into the IP header. Its 20 bytes contain the instructions to take it anywhere on the Internet.

The first byte is broken into two nibbles, the first 4 bits are the IP version which is currently still four. The next 4 bits are the header length, which is the number of 32 bit words in the header, 5 in this case. The second byte is the type of service to optimize for: minimize delay, maximize throughput, maximize reliability, or minimize monetary cost. The recommended value for ping is 0x00 which means normal service with no optimization. The third and fourth bytes are the 16 bit total length of the IP header plus the following data such as the ICMP message.

The next 4 bytes are for fragmented packets and since these packets are so small, this algorithm ignores fragmentation. The ninth byte is the time to live (TTL) flag and it sets the maximum number of routers a packet can pass through before it is discarded. This is important to keep the Internet from getting clogged with lost packets. The TTL flag is usually set to 32 or 64 and decremented by each router the packet passes through. The tenth byte is a protocol field which says what type of information the IP header is attached to.

Bytes 11 and 12 are called the header checksum, which is a 16 bit one's complement sum of the 20 header bytes. For implementation details check RFC 1071 which has a very good description and sample C code. Basically, a 16-bit one's complement sum is the 16 bit sum of 16 bit data where overflow carries into bit 16 are wrapped around and added to bit 0. The next 4 bytes are the source IP address and the last four bytes are the destination IP address.

The ICMP message follows with a type byte, code byte, and checksum word, see Figure 9. The type byte is 0 for a ping reply or 8 for a ping request. The code byte is zero in both cases and the checksum is again a one's complement sum. This time the checksum is the sum of the ICMP header plus the following data. The amount of data is the IP header total length minus the IP header length. In the case of a ping the originator stuffs in some data to see if it is properly echoed by the ping reply. This arbitrary data could just as well be your collected data or other information you wish to send. This algorithm responds to ping requests without echoing back any of the arbitrary data and causes some ping programs to report an error.

FIGURE 8: INTERNET PROTOCOL PACKET SHOWING MEMORY LOCATIONS IN RX BUFFER

Buffer Offset	bit 0		bit 8	bit 16		bit 24	bit 31
4	IP Version	Length	Type of Service	Total Length in Bytes			
8	Identification			Flags	Fragment Offset		
12	Time To Live (TTL)		Protocol	Header Checksum			
16	Source IP Address						
20	Destination IP Address						
24	ICMP, TCP, or UDP Header and Data						
	.						
	.						
	.						

FIGURE 9: A SAMPLE OF A PING WITH NO OPTIONAL DATA

Buffer Offset	bit 0		bit 8	bit 16	bit 24	bit 31	PPP Packet
							0000 : FF 03 00 21
4	Version 0100	Length 0101	Service 0000 0000	Total Length 00000000 00011100			0004 : 45 00 00 1C
8	Identification 1000 1000 0001 0000			Flags 010	Fragment Offset 00000 00000000		0008 : 88 10 40 00
12	TTL 0111 1111		Protocol 0000 0001	Header Checksum 00110011 10100111			000C : 7F 01 33 A7
16	Source IP Address 11001101 11001000 00101101 01111100						0010 : CD C8 2D 7C
20	Destination IP Address 11001111 10100001 01110101 01000011						0014 : CF A1 75 43
24	ICMP Type 0000 1000		ICMP Code 0000 0000	ICMP Checksum 11110111 11111110			0018 : 08 00 F7 FE
28	PING Identifier 00000000 00000001			PING Sequence Number 00000000 00000000			001C : 00 01 00 00
							0020 : 22 7C 7E

UDP DETAILS

UDP is the protocol required to transfer files with TFTP, convert host names to IP addresses with DNS, or status and event reporting with SNMP. Its simplicity and bandwidth efficiency make it an important part of some multimedia Internet protocols. The official specification is found in RFC 768.

This algorithm doesn't support UDP protocols but this section will give you a bit of background and make it easier for you to add it to the algorithm. First of all UDP is an unreliable protocol, not that it should be avoided, but rather that packets can get lost without warning and may require retransmission. It is deterministic in the sense that each packet, or timeout, triggers the next event without regard to what state the connection is in. This simplifies programming and makes debugging much easier.

The format of UDP is shown in Figure 10. There are 20 bytes of IP header, then 8 bytes of UDP header, and the UDP data. The first two 2 byte fields are the source and destination port numbers. The port numbers are important to identify what process gets the UDP data. An example is port 69 which is always used for TFTP.

The next two bytes are the UDP length, eight bytes of UDP header plus the length of the UDP data. This value is redundant because it can be calculated from the IP header by subtracting the header length from the total length.

The last 2 bytes of the UDP header are the 16 bit one's complement checksum of the pseudo header, the UDP header, and the UDP data. The pseudo header is not transmitted but the following 12 bytes are added to the checksum anyway. The 32 bit source and destination addresses, the 16-bit UDP address, and the 8-bit protocol field are extended to 16 bits to ensure that the UDP data is going to the correct IP address.

The checksum is optional and set to zero if not used. Since zero means no checksum, then a valid checksum that adds up to zero must be inverted to 0xFFFF. If the UDP data is an odd number of bytes, your 16 bit checksum routine will need to pretend that there is an extra byte 0x00 at the end.

The format of the UDP data will depend on which port you are connecting to and which protocol is using the data. A good example is the Trivial File Transfer Protocol (TFTP) which is well documented in RFC 1350.

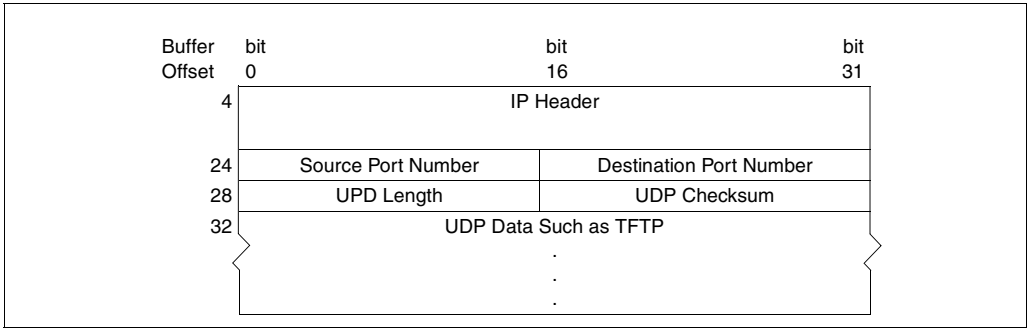
TCP DETAILS

TCP is the protocol required to transfer files with FTP, communicate by email with SMTP, login remotely with Telnet, or serve web pages with HTTP. The original specification is found in RFC 793; however, it has been improved by the Host Requirements RFCs 1122 and 1123.

This algorithm doesn't even pretend to support TCP packets because of the larger RAM and ROM requirements. Parts of the protocol may fit in a PICmicro MCUs with larger program memory size, so here is a little information for those brave enough to try. TCP is considered a reliable protocol because it hides lost and missing packets from the running applications: it tracks and retransmits them in the background.

This is really no different than UDP for the purpose of this algorithm since there is no distinction of software levels. In both cases the same process is responsible to retransmit missing packets. The difference is in the complexity and size of the packets. Another difference is that the other end of the connection is expecting this algorithm to keep track of packet timing, retransmit previous packets, and remember the state of multiple simultaneous connections.

FIGURE 10: THE UDP PACKET FORMAT



HARDWARE IMPLEMENTATION

This application note was designed for the PIC16C63A to demonstrate how compact the PPP connection algorithm can be shrunk. The algorithm uses 151 of the 192 file registers and 2.2k of the 4k ROM and only six I/O pins. There is still plenty of space for your own code and the code is portable enough to move it into a smaller or larger PICmicro MCU. The 4 MHz crystal is fast enough and could be slowed down, unless you need a faster modem or run additional CPU intensive tasks.

The modem is a Ceremtek CH1786LC, running at 2400 baud, but with packet sizes under 50 bytes speed is not much of an issue. For higher traffic connections or large data transfers you may want to upgrade to the larger but still pin-compatible 14.4 kilobaud CH1794. Be sure to check with the modem manufacturer what external circuitry is required before connecting to the telephone line. Your circuit must be designed and tested to meet the telecom standards of the country in which you wish to use it.

Only the telephone line, power, serial transmit, receive, and DTR line need to be connected to use this modem. The DTR line must be tied low for the modem to operate properly. The modem can be very sensitive to power supply noise so be sure to keep it close to a bypass capacitor. You could also change the software a little and replace the modem with an RS232 driver to go directly to the server's serial port.

One desirable characteristic of this modem is that it draws a maximum of 50 mA. Since it is only on for brief periods and the entire circuit never draws more than a total of 65 mA, we can easily power it off a 9V battery. A typical 9V alkaline battery with a 560mA hour rating would give us about 9 hours of power. The modem off hook to ping time is under a minute, so if we just send one ping and hang up, the battery would last for more than 500 pings. This works out to be about a year and a half at one ping per day.

This requires the power supply to turn off completely after the ping is successful. The power supply design uses a NPN transistor to turn on a PNP transistor which supplies the current to the voltage regulator. The NPN transistor can be turned on by the processor or by a momentary switch. When the momentary switch closes it turns on the power, and as the microcontroller initializes it too turns on the NPN transistor. By this time the user has released the switch and the microcontroller keeps the power on. The switch could be almost anything like a magnetic burglar sensor or even a thermostat. When the ping is complete it releases the power and turns itself off. If you need to ping the device, just keep the manual switch closed until you want the power off. With the switch released, it will power down after the first successful ping or an automatic timeout if there is no modem connection, password fails, or no ping replies.

The PNP transistor is also a benefit to reducing power consumption because no voltage drop is lost to a reverse protection diode. You may also upgrade to a low dropout (LDO) voltage regulator to get a little extra life out of the battery before the 5 volt regulator stops regulating. Choose a regulator that includes a power switch, or change the PNP to a MOSFET to reduce the current draw by one mA and add another two percent to the battery life. Rather than improving efficiency with a DC-DC converter, I would choose a lower-voltage battery pack with a flat discharge curve that barely maintains the minimum LDO regulating voltage. If the device will be inaccessible or needs a long shelf life, then go with a lithium-ion battery pack.

There are three LEDs: one indicates the modem status and the other two indicate serial data transmitted and received. The modem status LED is off while the software initalizes the modem, flashes quickly while dialing, and goes on steady when connected. If it goes off after flashing then it didn't connect and it will try again in a couple seconds. After connecting and negotiating PPP the status light will go off for a second and then flash out its 32 bit IP address. A long flash is a one and a short flash is a zero. Write down each bit as it flashes and then convert the binary to hexadecimal. Make it easier by grouping the bits in fours, each a hexadecimal character. Insert three decimals spaced every 8 bits, convert the four numbers to decimal, and you have your IP address, see Figure 11. This address is usually dynamically assigned resulting in a different address every time it logs on the Internet.

Note:	For first-time developers of PICmicro MCUs, using the Microchip PICDEM Demonstration boards (DV163002) may be useful.

FIGURE 11: CALCULATION OF IP ADDRESS FROM LED PULSES

Record long flashes as 1 bit.
Record short flashes as 0 bit.
There will be a pause every eight flashes.
Example:

	1100	1101		1100	1000		0010	1101		0100	1010				
Hexadecimal:	C	D	•	C	8	•	2	D	•	4	A				
Decimal:	205		•	200		•	45		•	74					

SOFTWARE IMPLEMENTATION

The software is written in C to keep it portable. This way the algorithm can be developed on a PC and tested with a variety of low-cost compilers and debuggers or simply print all relevant data to the screen. Then, just press PrintScreen or use DOS to pipe the screen output to a file for analysis. To compile for a PC, replace the serial functions with COM port routines and use the PC tick counter at address '0040:006C' instead of TMR0.

There are a number of excellent C compilers for the PICmicro MCU. This code was started with the free compiler CC5X from B. Knudsen Data in Norway and completed with the PICmicro MCU C Compiler from HiTech in Australia. The code shown will compile with the HiTech demo available at http://www.htsoft.com. It should work with all the other C compilers for the PICmicro MCU if you do all compiler specific modifications required to the code.

The code consists of a main routine that does the two main tasks of modem control scripting and the protocol state machine. There are a couple of support routines for calculating the CRC checksums, creating packets, checking configuration options, and controlling the modem.

When you press the power switch, the microcontroller powers up, does a short time delay loop, and asserts RB3 to keep the power on. The 250 millisecond time delay is meant to prevent false triggering. As long as either the user is pressing the power button or RB3 is asserted the power will stay on. The software will release RB3 to turn off the power after it successfully pings a remote host, times out trying, or fails 20 dial attempts. If the power button is still pressed, the software continues to dial or attempt more pings until the button is released. For example, if you want to ping it from another computer you will have to hold in the button until you complete your ping tests.

The software will attempt to phone the number programmed in the source code up to 20 times at about 30-second intervals. When it connects it tries going from a script login to a PPP login by sending a PPP packet instead of a User ID. If that fails, it falls back to a script login; otherwise it goes into the main loop of the algorithm. The protocol state machine loop does all the serial I/O, packet processing, packet creation, and timing to negotiate PPP and complete a ping.

The state machine starts in state 0. When the Internet server acknowledges the LCP configuration packet state, bit 0 is set. When the algorithm acknowledges the server's LCP configuration, bit 1 is set. As long as bit 0 is clear, the algorithm will send an LCP request once every second. When both bits are set the algorithm moves into state 4.

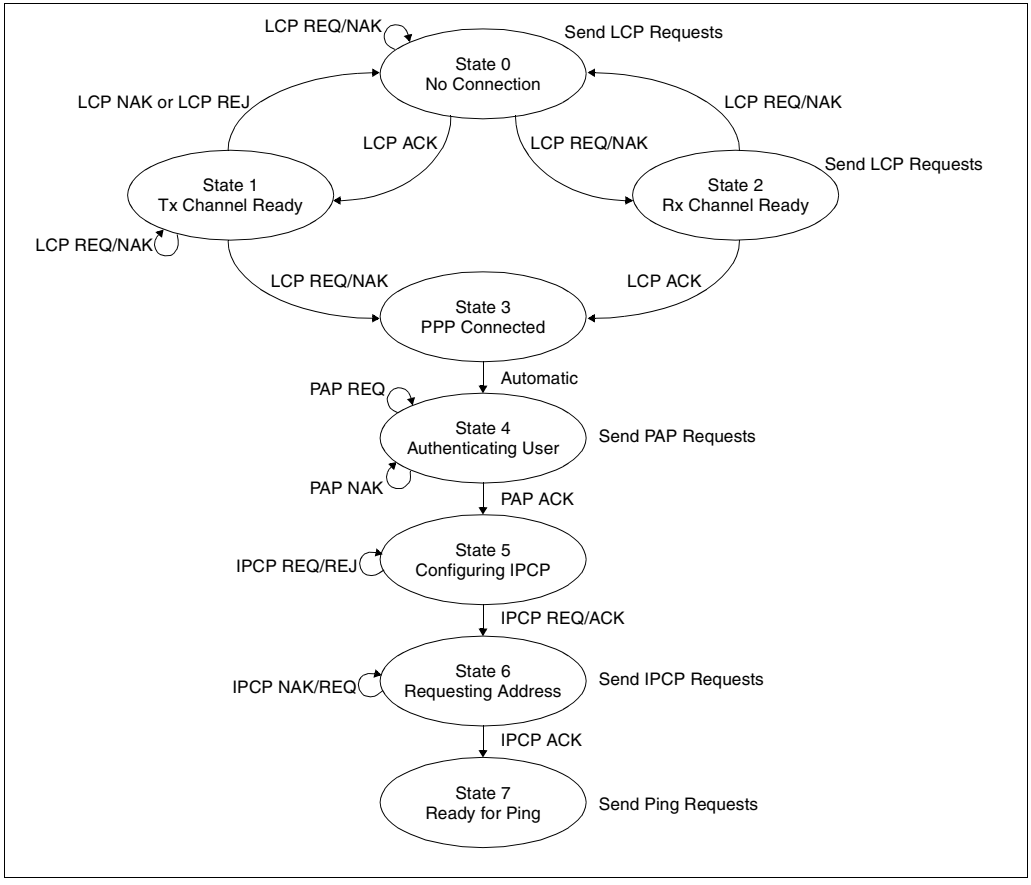
In state four a PAP request with the User ID and password is sent once a second. The acknowledgement of the password moves the algorithm into state five. When the algorithm acknowledges the server's IPCP options it moves into state six. In state six the algorithm requests IP address 0.0.0.0 once a second. The server should reply with a NAK packet containing the correct address to move the algorithm into it final state 7. Here it flashes out the IP address on the status LED and then pings the hardcoded host IP address every 30 seconds. After the first good ping reply, it turns off the power unless the power button is still pressed.

The `MakePacket` routine creates an outgoing packet in the transmit buffer. Every loop of the state machine checks if the serial transmitter is ready for another character. If the transmit buffer is empty, it sends the next character. On the last character it marks the buffer empty and sends an extra 0x7E to mark the end of the packet.

Every loop of the state machine also checks the serial receiver for characters from the modem or Internet server. Characters that are sent using the previously described } escape sequence are immediately converted back to the original character. The CRC checksum is also calculated as the bytes come in so that packets longer than the buffer can still pass the CRC.

The `OptionTest` routine is used to test the receive buffer for whatever options the server is requesting. It takes a 16-bit option parameter, where each bit represents an option from 1 to 16 - with 16 being the MSB. If a bit is set, then its corresponding option can be accepted. If the server requests options that are not allowed by the option parameter, then the subroutine returns a zero and deletes the options that were allowed. This way a REJ packet can be sent to tell the server which options to drop. If it is an LCP packet with option three set to CHAP then the subroutine returns a one and deletes the options that passed. This way a NAK packet can be sent to tell the server to switch to PAP. In all other cases the subroutine returns a value greater than one and leaves the receive buffer unchanged.

FIGURE 12: PPP NEGOTIATION STATE MACHINE



CONCLUSION

This algorithm is a little taste of what is possible with a PICmicro MCU, you will likely use this information as a basis for even more powerful Internet applications. Just remember to only make small changes to working code and test them before making the next change.

This information is quite technical, so don't give up if you are not already TCP/IP savvy. Remember that all the so called experts had to learn it at some time too. Read a book like TCP/IP Illustrated Volume 1 by Richard Stevens and the referenced RFCs, then compile the code and analyze lots of packets. Start your experimenting slowly with a relatively easy task like adding support for replying to Traceroute requests. Here's a hint: test the TTL on valid IP packets received to trigger sending an ICMP error packet.

This tutorial was meant to encourage the development of tiny Internet interfaces and not to replace or override the established Internet standards documents.

The prototype does what I needed it to do but there are many areas in which it could be improved upon, such as size, speed, power requirements, RAM usage, supported protocols, and more universal PPP negotiations. The possibilities are only limited by your imagination and creativity in overcoming the obstacles.

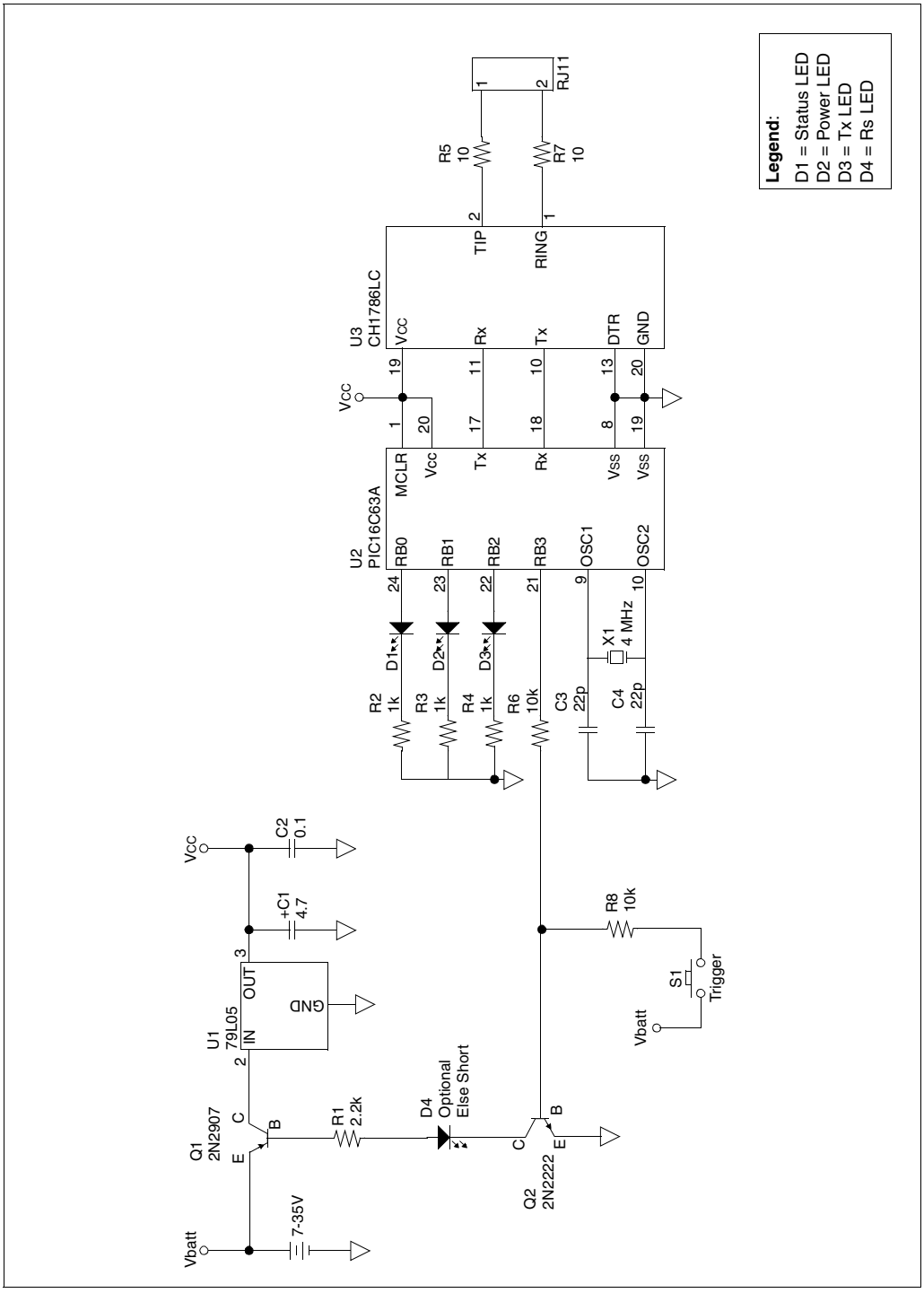
TABLE 5: REFERENCES

W.R. Stevens, *TCP/IP Illustrated*, Vol. 1, Addison Wesley, Reading, MA, 1994
James Carlson, PPP Design and Debugging, Addison Wesley, Reading, MA, 1997
RFC 0768 User Datagram Protocol. J. Postel. Aug-28-1980.
RFC 0791 Internet Protocol. J. Postel. Sep-01-1981.
RFC 0792 Internet Control Message Protocol. J. Postel. Sep-01-1981.
RFC 0793 Transmission Control Protocol. J. Postel. Sep-01-1981.
RFC 0867 Daytime Protocol. J. Postel. May-01-1983.
RFC 0950 Internet Standard Subnetting Procedure. J.C. Mogul, J. Postel. Aug-01-1985.
RFC 1055 Nonstandard for transmission of IP datagrams over serial lines: SLIP. J.L. Romkey. Jun-01-1988.
RFC 1071 Computing the Internet checksum. R.T. Braden, D.A. Borman, C. Partridge. Sep-01-1988.
RFC 1122 Requirements for Internet hosts - communication layers. R.T. Braden. Oct-01-1989.
RFC 1123 Requirements for Internet hosts - application and support. R.T. Braden. Oct-01-1989.
RFC 1144 Compressing TCP/IP headers for low-speed serial links. V. Jacobson. Feb-01-1990.
RFC 1157 Simple Network Management Protocol J.D. Case, M. Fedor, M.L. Schoffstall, C. Davin. May-01-1990.
RFC 1332 The PPP Internet Protocol Control Protocol (IPCP). G. McGregor.May 1992.
RFC 1334 PPP Authentication Protocols. B. Lloyd, W. Simpson. October 1992.
RFC 1350 The TFTP Protocol (Revision 2). K. Sollins. July 1992.
RFC 1547 Requirements for an Internet Standard Point-to-Point Protocol. D. Perkins. December 1993.
RFC 1570 PPP LCP Extensions. W. Simpson. January 1994.
RFC 1661 The Point-to-Point Protocol (PPP). W. Simpson, Editor. July 1994.
RFC 1662 PPP in HDLC-like Framing. W. Simpson, Editor. July 1994.
RFC 1663 PPP Reliable Transmission. D. Rand. July 1994.
RFC 1700 Assigned Numbers. J. Reynolds, J. Postel. October 1994.
RFC 1962 The PPP Compression Control Protocol (CCP). D. Rand. June 1996.
RFC 1989 PPP Link Quality Monitoring. W. Simpson. August 1996.
RFC 1994 PPP Challenge Handshake Authentication Protocol (CHAP). W. Simpson. August 1996.

RFCs available online from sites like: http://www.cis.ohio-state.edu/hypertext/information/rfc.html

Assigned PPP Numbers: ftp://ftp.isi.edu/in-notes/iana/assignments/ppp-numbers

FIGURE 13: SCHEMATIC



APPENDIX A: SOURCE CODE

SOFTWARE LICENSE AGREEMENT

The software supplied herewith by Microchip Technology Incorporated (the “Company”) for its PICmicro® Microcontroller is intended and supplied to you, the Company’s customer, for use solely and exclusively on Microchip PICmicro Microcontroller products.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

APPENDIX A: SOURCE CODE

```
////////////////////////////////////////////////////////////////////////////////
//
//    PING.C    version 1.10    July 29/99     (C)opyright by Microchip Technology Inc.
//
////////////////////////////////////////////////////////////////////////////////
//
// For more documentation read the Microchip Application Note 724
// This code is ready to compile with the HiTech C compiler demo for the PIC16C63A.
//
// You will need these additional things to make this code work:
//
//    - the simple hardware described in application note
//
//    - an Internet account with PPP dialup access (not compatible with all ISPs)
//
//    - replace 5551234 with your ISP's phone number in the line like this
//         if (sendwait("5551234\r","NNECT",3000)) {
//
//    - replace userid with your account userid in the line like this:
//         if (sendwait("userid\r","word:",200))
//
//    - replace password with your account password in the line like this:
//         if (sendwait("password\r","tion:",1000))
//
//    - replace the entire string in the line like this:
//         MakePacket(PAP,REQ,number,"\x14\x06userid\x08password");
//
//         C converts the \x## in the string to a character with that ASCII value
//         ## is a hexadecimal value, so the following character cannot be
//         if the next character is 0-9 or A-F or a-f then it will confuse the compiler
//         the solution is to convert the next characters to \x## until a non hex char
//         if in doubt look at the assembly output from the compiler
//         replace the userid with yours and the \x06 with your userid length
//         replace the password with yours and the \x08 with your password length
//         replace the first value in the string, it must be the string length plus 4
//
// Once login is working you should also change the IP address of the Internet host to ping
// if you can not ping 207.161.117.67 with your PC this code will not work either
//    It is CF.A1.75.43, the characters 2 to 5, in the string in the line like this:
//       MakePacket(IP,0,1,"\x10\xCF\xA1\x75\x43\x8\x0\xF7\xFE\x0\x1\x0\x0");
//    Convert the address you want to hexadecimal and replace the four values.
//
// Make sure the power-on reset and brownout detect config bits are enabled
//
////////////////////////////////////////////////////////////////////////////////

// Defines for Internet constants
#define  REQ        1                   // Request options list for PPP negotiations
#define  ACK        2                   // Acknowledge options list for PPP negotiations
#define  NAK        3                   // Not acknowledged options list for PPP negotiations
#define  REJ        4                   // Reject options list for PPP negotiations
#define  TERM       5                   // Termination packet for LCP to close connection
#define  IP         0x0021              // Internet Protocol packet
#define  IPCP       0x8021              // Internet Protocol Configure Protocol packet
#define  CCP        0x80FD              // Compression Configure Protocol packet
#define  LCP        0xC021              // Link Configure Protocol packet
#define  PAP        0xC023              // Password Authentication Protocol packet

#define  MaxRx      46                  // Maximum size of receive buffer
#define  MaxTx      46                  // Maximum size of transmit buffer

unsigned char addr1, addr2, addr3, addr4;// Assigned IP address
unsigned int rx_ptr, tx_ptr, tx_end;     // pointers into buffers
unsigned int checksum1, checksum2;       // Rx and Tx checksums
```

```
unsigned char number;                    // Unique packet id

#include <pic1663.h>                     // Defines specific to this processor

#define serial_init()          RCSTA=0x90;TXSTA=0x24;SPBRG=103// Set up serial port
#define serial_tx_ready()      TXIF                    // Transmitter empty
#define serial_send(a)         TXREG=a                 // Transmit char a
#define serial_rx_ready()      RCIF                    // Receiver full
#define serial_get()           RCREG                   // Receive char
#define serial_error()         OERR                    // USART error
#define serial_fix()           {CREN=0;CREN=1;}        // Clear error

unsigned int TIME;                                     // 10 milliseconds counter
#define TIME_SET(a) TIME=a                             // Set 10 millisecond counter to value 'a'
bank1 unsigned char tx_str[MaxRx+1];                   // Transmitter buffer
bank1 unsigned char rx_str[MaxTx+1];                   // Receiver buffer

// Process all the interrupts in the PIC here
static void interrupt isr(void) {
   if (T0IF) {                           // Timer overflow interrupt?
      TMR0 = 100;                        // Set to overflow again in 10ms @ 4MHz
      T0IF = 0;                          // Clear overflow interrupt flag
      TIME++;                            // Increment 10 ms counter
   }
}

// Add next character to the CRC checksum for PPP packets
unsigned int calc(unsigned int c) {
   char i;                               // Just a loop index
   c &= 0xFF;                            // Only calculate CRC on low byte
   for (i=0;i<8;i++) {                   // Loop eight times, once for each bit
      if (c&1) {                         // Is bit high?
         c /= 2;                         // Position for next bit
         c ^= 0x8408;                    // Toggle the feedback bits
      } else c /= 2;                     // Just position for next bit
   }                                     // This routine would be best optimized in assembly
   return c;                             // Return the 16 bit checksum
}

// Add character to the new packet
void add(unsigned char c) {
   checksum2 = calc(c^checksum2) ^ (checksum2/256);  // Add CRC from this char to running total
   tx_str[tx_ptr] = c;                               // Store character in the transmit buffer
   tx_ptr++;                                         // Point to next empty spot in buffer
}

// Create packet of type, code, length, and data string specified
//   packet is the type, like LCP or IP
//   code is the LCP type of packet like REQ, not used for IP packets
//   num is the packet ID for LCP, or the IP data type for IP packets
//   *str is the packet data to be added after the header
//   returns the packet as a string in tx_str
void MakePacket(unsigned int packet, unsigned char code, unsigned char num, const unsigned char
*str) {
   unsigned int length;                        // Just a dual use temp variable
   tx_ptr = 1;                                 // Point to second character in transmit buffer
   tx_str[0] = ' ';                            // Set first character to a space for now
   checksum2 = 0xFFFF;                         // Initialize checksum
   add(0xFF);                                  // Insert PPP header OxFF
   add(3);                                     // Insert PPP header 0x03
   add(packet/256);                            // Insert high byte of protocol field
   add(packet&255);                            // Insert low byte of protocol field
   if (packet==IP) {                           // If Internet Protocol
      add(0x45);                               // Insert header version and length
      add(0);                                  // Insert type of service
      add(0);                                  // Insert total packet length high byte
```

```
      add((*str)+12);                          // Insert total packet length low byte
      add(0x88);                               // Insert identification high byte
      add(0x10);                               // Insert identification low byte
      add(0x40);                               // Insert flags and fragment offset
      add(0);                                  // Insert rest of fragment offset
      add(127);                                // Insert time to live countdown
      add(num);                                // insert the protocol field
      length = 0x45+0x88+0x40+127+addr1+addr3+str[1]+str[3];  // high byte checksum
      packet = *str + 12 + 0x10 + num + addr2 + addr4 + str[2] + str[4];
                                               // low byte checksum
      packet += length/256;                    // make 1's complement
      length = (length&255) + packet/256;      // by adding low carry to high byte
      packet = (packet&255) + length/256;      // and adding high carry to low byte
      length += packet/256;                    // fix new adding carries
      add(~length);                            // Insert 1's complement checksum high byte
      add(~packet);                            // Insert 1's complement checksum low byte
      add(addr1);                              // Insert the 4 bytes of this login's IP address
      add(addr2);
      add(addr3);
      add(addr4);
      length = *str - 4;                       // save the number of following data bytes
      str++;                                   // point to the first data byte
   } else {
      add(code);                               // Insert packet type, like REQ or NAK
      add(num);                                // Insert packet ID number
      add(0);                                  // Insert most significant byte of length
      length = *str - 3;                       // point to the first data byte
   }
   while (length) {                            // copy the whole string into packet
      length--;                                // decrement packet length
      add(*str);                               // add current character to packet
      str++;                                   // point to next character
   }
   length = ~checksum2;                        // invert the checksum
   add(length&255);                            // Insert checksum msb
   add(length/256);                            // Insert checksum lsb
   tx_end=tx_ptr;                              // Set end of buffer marker to end of packet
   tx_ptr = 0;                                 // Point to the beginning of the packet
}

// Test the option list in packet for valid passwords
//   option is a 16 bit field, where a high accepts the option one greater than the bit #
//   returns 2 for LCP NAK, 1 is only correct fields found, and zero means bad options
//   return also modifies RX_STR to list unacceptable options if NAK or REJ required
unsigned char TestOptions(unsigned int option){
   unsigned int size;                          // size is length of option string
   unsigned ptr1 = 8,                          // ptr1 points data insert location
            ptr2 = 8;                          // ptr2 points to data origin
   char pass = 3;                              // pass is the return value
   size = rx_str[7]+4;                         // size if length of packet
   if (size>MaxRx) size=MaxRx;                 // truncate packet if larger than buffer
   while (ptr1<size) {                         // scan options in receiver buffer
      if (rx_str[ptr1]==3 && rx_str[ptr1+2]!=0x80  && rx_str[2]==0xc2)
         pass&=0xfd;                           // found a CHAP request, mark for NAK
      if (!((1<<(rx_str[ptr1]-1))&option))
         pass=0;                               // found illegal options, mark for REJ
      ptr1 += rx_str[ptr1+1];                  // point to start of next option
   }
   if (!(pass&2)) {                            // If marked for NAK or REJ
      if (pass&1) {                            // save state for NAK
         option=0xfffb;
      }
      for (ptr1=8; ptr1<size;) {
         if (!((1<<(rx_str[ptr1]-1))&option)) {// if illegal option
            for (pass=rx_str[ptr1+1]; ptr1<size && pass; ptr1++) {  // move option
               rx_str[ptr2]=rx_str[ptr1];      // move current byte to new storage
```

```
               ptr2++;                          // increment storage pointer
               pass--;                          // decrement number of characters
            }
         } else {
            ptr1+=rx_str[ptr1+1];               // point to next option
         }
      }
      rx_str[7] = ptr2-4;                       // save new option string length
      pass=0;                                   // restore state for REJ
      if (option==0xfffb) pass=1;               // restore state for NAK
   }
   return pass;
}

// Send a string and loop until wait string arrives or it times out
//  send is the string to transmit
//  wait is the string to wait for
//  timeout is in multiples of 10 milliseconds
//  addr1 is used to control the status LED, 0=off, 1=flash, 2=on
//  returns 0 if timeout, returns 1 if wait string is matched
char sendwait(const char *send, const char *wait, unsigned int timeout) {
   addr2=addr3=0;
   for (TIME_SET(0); TIME<timeout; ) {            // loop until time runs out
      if (!addr1) PORTB&=0xFB;                    // if addr1=0 turn off status LED
      else if (addr1==1) {                        // if addr1=1 flash status LED
         if (TIME&4) PORTB&=0xFB;                 // flash period is 8 x 10ms
         else PORTB|=4;
      } else PORTB|=4;                            // if addr1>1 turn on status LED
      if (serial_rx_ready()) {                    // is there an incoming character
         PORTB|=1;                                // turn on the Rx LED
         addr4 = serial_get();                    // get character
         if (serial_error()) serial_fix();        // clear serial errors
         if (wait[addr2]==addr4) addr2++;         // does char match wait string
         else addr2=0;                            // otherwise reset match pointer
         PORTB&=0xFE;                             // turn off the Rx LED
         if (!wait[addr2]) return 1;              // finished if string matches
      } else if (send[addr3] && (serial_tx_ready())) { // if char to send and Tx ready
         if (send[addr3]=='|') {                  // if pause character
            if (TIME>100) {                       // has 1 second expired yet?
               TIME_SET(0);                       // if yes clear timer
               addr3++;                           // and point to next character
            }
         } else {
            PORTB|=2;                             // turn on Tx LED
            TIME_SET(0);                          // clear timer, timeout starts after last char
            serial_send(send[addr3]);             // send the character
            addr3++;                              // point to next char in tx string
         }
         PORTB&=0xFD;                             // turn off Tx LED
         if (!send[addr3] && !(*wait))
            return 1;                             // done if end of string and no wait string
      }
   }
   return 0;                                      // return with 0 to indicate timeout
}

void flash(void) {                                // flash all LEDs if catastrophic failure
   for (TIME_SET(0);;) {
      if (TIME&8) PORTB|=0x07;                    // flash period is 16 x 10ms
      else PORTB&=0xF8;
      if (TIME>3000) PORTB&=0xF7;                 // after 30 seconds turn off the power
   }
}

void pulse(unsigned char data) {                  // pulse Status LED with IP address
   TIME_SET(0);
```

```
   for(number=0;number<9;) {                    // pulse out 8 address bits and a blank
      if (TIME<100) PORTB&=0xFB;                // turn off Status LED between bits
      else if (number<8) PORTB|=4;              // start each address bit here
      if (TIME>200 || (!(data&0x80) && TIME>120)) { // end of bit?
         TIME_SET(0);                           // yes, then restart timer for next bit
         number++;                              // increment bit counter
         data<<=1;                              // position address to send next bit
      }
   }
}

// The main loop, login script, PPP state machine, and ping transponder
void main(void) {
   signed int c;                          // serial character received
   unsigned int packet = 0;               // Type of the last received packet, reused as temp
   unsigned char state = 0;               // PPP negotiation state, from dialing=0 to done=6
   unsigned char extended = 0;            // flag if last character was an escape sequence

   PORTA=0;
   PORTB=0;                               // Turn off power supply, turn off LEDs
   PORTC=0;
   TRISA=0;                               // Turn all I/O into outputs
   TRISB=0x00;
   TRISC=0xC0;
   OPTION=0x85;                           // Set up TIMER 0 for millisecond counting
   INTCON=0xA0;

   serial_init();                         // Initialize serial port to 2400 baud format N81
   TIME_SET(0);
   while (TIME<25);                       // 250 millisecond delay to prevent false power up
   PORTB=8;                               // Turn on the power so user can release power button

   for(number=1;;number++) {              // Redial indefinitely every 30 seconds
      if (number==10) PORTB&=0xF7;        // Turn off power if dialing fails
      addr1=0;                            // Set flag to keep the Status LED off
      if(!sendwait("|+++|\rath\r|atz\r|at&fs11=55\r|atdt","atdt",3000))// Init modem
         flash();
      addr1=1;                            // Set flag to flash Status LED
      // Modify this line with your ISP phone number
      if (sendwait("5551234\r","NNECT",3000)) {
         addr1=2;                         // Set flag to keep the Status LED on
         if (sendwait("",": ",1000))      // Wait for user id prompt
            if (sendwait("\x7e\xff\x7d\x23\x08\x08\x08\x08","~~",1000))
               break;                     // Start PPP
            else {                        // Fallback to script if required
               if (sendwait("userid\r","word:",200))// Modify these lines as described
                  if (sendwait("password\r","tion:",1000))
                     if (!sendwait("ppp\r","IP address",200))
                       // Modify is start PPP command is not ppp or 2
                       sendwait("2\r","IP address",200);
            }
         break;
      }
   }

   // State machine loop until successful ping or PPP negotiation timeout
   for (TIME_SET(0);;) {
      if (TIME>7000 || number>20) PORTB&=0xF7;
      if (serial_rx_ready()) {            // Incoming character?
         PORTB ^=1;                       // Turn on Rx LED
         c = serial_get();                // get the character
         if (serial_error()) serial_fix();// clear Rx errors
         if (c == 0x7E) {                 // start or end of a packet
            if (rx_ptr && (checksum1==0xF0B8))
               packet = rx_str[2]*256 + rx_str[3]; // if CRC passes accept packet
            extended &= 0x7E;             // clear escape character flag
```

```
            rx_ptr = 0;                  // get ready for next packet
            checksum1 = 0xFFFF;          // start new checksum
         } else if (c == 0x7D) {         // if tilde character set escape flag
            extended |= 1;
         } else {
            if (extended&1) {            // if escape flag
               c ^= 0x20;                // recover next character
               extended &= 0xFE;         // clear Rx escape flag
            }
            if (rx_ptr==0 && c!=0xff) rx_str[rx_ptr++] = 0xff; // uncompress PPP header
            if (rx_ptr==1 && c!=3) rx_str[rx_ptr++] = 3;
            if (rx_ptr==2 && (c&1)) rx_str[rx_ptr++] = 0;
            rx_str[rx_ptr++] = c;        // insert character in buffer
            if (rx_ptr>MaxRx) rx_ptr = MaxRx;// Inc pointer up to end of buffer
            checksum1 = calc(c^checksum1) ^ (checksum1/256); // calculate CRC checksum
         }
         PORTB&=0xFE;                    // turn off Status LED
      } else if (tx_end && (serial_tx_ready())) { // Data to send and Tx empty?
         PORTB|=2;                       // turn on Tx LED
         c = tx_str[tx_ptr];             // get character from buffer
         if (tx_ptr==tx_end) {           // was it the last character
            tx_end=0;                    // mark buffer empty
            c='~';                       // send tilde character last
            PORTB&=0xFD;                 // turn off Tx LED
         } else if (extended&2) {        // sending escape sequence?
            c^=0x20;                     // yes then convert character
            extended &= 0xFD;            // clear Tx escape flag
            tx_ptr++;                    // point to next char
         } else if (c<0x20 || c==0x7D || c==0x7E) { // if escape sequence required?
            extended |= 2;               // set Tx escape flag
            c = 0x7D;                    // send escape character
         } else {
           if (!tx_ptr) c='~';           // send ~ if first character of packet
           tx_ptr++;
         }
         serial_send(c);                 // Put character in transmitter
      }

      if (packet == LCP) {
         switch (rx_str[4]) {            // Switch on packet type
            case REQ:
               state &= 0xfd;            // clear remote ready state bit
               if (c=TestOptions(0x00c6)) {// is option request list OK?
                  if (c>1) {
                     c = ACK;            // ACK packet
                     if (state<3) state |= 2;// set remote ready state bit
                  } else {
                     rx_str[10]=0xc0;    // else NAK password authentication
                     c = NAK;
                  }
               } else {                  // else REJ bad options
                  c = REJ;
               }
               TIME_SET(0);
               MakePacket(LCP,c,rx_str[5],rx_str+7); // create LCP packet from Rx buffer
               break;
            case ACK:
               if (rx_str[5]!=number) break;// does reply id match the request
               if (state<3) state |= 1;  // Set the local ready flag
               break;
            case NAK:
               state &= 0xfe;            // Clear the local ready flag
               break;
            case REJ:
               state &= 0xfe;            // Clear the local ready flag
               break;
```

```
      case TERM:
         break;
   }
   if (state==3) state = 4;         // When both ends ready, go to state 4
} else if (packet == PAP) {
   switch (rx_str[4]) {             // Switch on packet type
      case REQ:
         break;                     // Ignore incoming PAP REQ
      case ACK:
         state = 5;                 // PAP ack means this state is done
         break;
      case NAK:
         break;                     // Ignore incoming PAP NAK
   }
} else if (packet == IPCP) {
   switch (rx_str[4]) {             // Switch on packet type
      case REQ:
         if (TestOptions(0x0004)) {// move to next state on ACK
            c = ACK;
            state = 6;
         } else {                   // otherwise reject bad options
            c = REJ;
         }
         MakePacket(IPCP,c,rx_str[5],rx_str+7);
         break;                     // Create IPCP packet from Rx buffer
      case ACK:
         if (rx_str[5]==number) {   // If IPCP response id matches request id
            state = 7;              // Move into final state
            pulse(addr1);           // Pulse Status LED to show the
            pulse(addr2);           // IP address
            pulse(addr3);
            pulse(addr4);
            PORTB|=4;               // Turn on Status LED after pulsing
            TIME_SET(5800);         // Move timer ahead for quicker PING
         }
         break;
      case NAK:                     // This is where we get our address
         addr1 = rx_str[10];
         addr2 = rx_str[11];        // Store address for use in IP packets
         addr3 = rx_str[12];
         addr4 = rx_str[13];
         MakePacket(IPCP,REQ,rx_str[5],rx_str+7);
         break;                     // Make IPCP packet from Rx buffer
      case REJ:
         break;                     // Ignore incoming IPCP REJ
      case TERM:
         break;                     // Ignore incoming IPCP TERM
   }
} else if (packet == IP) {
   if (state<7 || (rx_str[19]==addr4 && rx_str[18]==addr3 &&
                   rx_str[17]==addr2 && rx_str[16]==addr1)) {
            // ignore echoed packets from our address or before we reach state 7
            // may power down here because echoes are good indications that modem
            // connection hung-up
            // This would be a good place to insert a traceroute test and
            // response
   } else if (rx_str[13]==1) {      // IP packet with ICMP payload
      if (rx_str[24]==8) {          // Received PING request
         rx_str[20]=rx_str[16];     // Copy 4 origin address bytes to
                                    // destination address
         rx_str[21]=rx_str[17];
         rx_str[22]=rx_str[18];
         rx_str[23]=rx_str[19];
         rx_str[19]=16;             // Length of IP address(4) + ping protocol(8) + 4
         rx_str[24]=0;              // Change received ping request(8) to ping reply(0)
         packet = rx_str[28]+rx_str[30];// Calculate 1's comp checksum
```

```
            rx_str[26] = packet&255;
            rx_str[27] = packet/256;
            packet = rx_str[27]+rx_str[29]+rx_str[31];
            rx_str[27] = packet&255;
            packet = packet/256 + rx_str[26];
            rx_str[26] = packet&255;
            rx_str[27] += packet/256;
            rx_str[26] = ~rx_str[26]; // Invert the checksum bits
            rx_str[27] = ~rx_str[27];
            MakePacket(IP,0,1,rx_str+19);// Make IP packet from modified Rx buffer
         } else if (rx_str[24]==0) {   // Received PING reply
            if ((rx_str[28]|rx_str[30]|rx_str[31])+rx_str[29]==1)
               PORTB&=0xF7;           // Turn off the power after successful ping
         }
      }
   } else if (packet == CCP) {
      switch (rx_str[4]) {            // If CCP response id matches request id
         case REQ:
            c = REJ;
            if (TestOptions(0x0004)) c = ACK; // ACK option 3 only, REJ anything else
            MakePacket(CCP,c,rx_str[5],rx_str+7);// Create CCP ACK or REJ packet
                                       // from Rx buffer
      }
   } else if (packet) {               // Ignore any other received packet types
   } else if (!tx_end && (state==0 || state==2) && TIME>100) {
                                       // Once a second try negotiating LCP
      number++;                        // Increment Id to make packets unique
      TIME_SET(0);                     // Reset timer
      MakePacket(LCP,REQ,number,"\x0E\x02\x06\x00\x0A\x00\x00\x07\x02\x08\x02");
                                       // Request LCP options 2,7,8
   } else if (!tx_end && state == 4 && TIME>100) {
                                       // Once a second try negotiating password
      TIME_SET(0);                     // Reset timer
      number++;
      // format like printf("%c%c%s%c%s",strlen(name)+strlen(password)+6,
      //             strlen(name),name,strlen(password),password);
      // Modify this line as described above
      MakePacket(PAP,REQ,number,"\x14\x06\xuserid\x08password");
   } else if (!tx_end && state == 6 && TIME>100) {
                                       // Once a second try negotiating IPCP
      number++;                        // Increment Id to make packets unique
      TIME_SET(0);                     // Reset timer
      MakePacket(IPCP,REQ,number,"\xA\x3\x6\x0\x0\x0\x0");
                                       // Request IPCP option 3 with addr 0.0.0.0
   } else if (!tx_end && state == 7 && TIME>3000) { // Every 30 seconds do a ping
      TIME_SET(0);                     // Reset timer
      number++;                        // Increment ping count
      MakePacket(IP,0,1,"\x10\xCF\xA1\x75\x43\x8\x0\xF7\xFE\x0\x1\x0\x0");
                                       // Ping 207.161.117.67
   }
   packet = 0;                         // Indicate that packet is processed
  }
}
```

PIC17CXXX to PIC18CXXX Migration

Author: *Mark Palmer*
Microchip Technology Inc.

INTRODUCTION

The specification of the PIC18CXXX Architecture was done with several goals. One of the most important of these goals was code compatibility with existing PICmicro® families. This goal eases the migration from one product family to the PIC18CXXX family.

For customers that are designing a new application that is based on an existing PICmicro device, but require added functionality (memory space, performance, peripheral features, ...), having source code compatibility is very useful (eases the development).

This application note looks at what may need to be addressed when migrating an application from a PIC17CXXX device to a PIC18CXXX device. It will not address the details of layout issues due to the different pinouts between these two families.

So looking at the issues for a code conversion, the following few points need to be inspected:

- Module Differences
- Memory Map Differences
  - Program Memory Map
  - Data Memory Map
- Instruction Execution Differences
- Architectural Modifications (such as Table Read and Table Write implementation)

Like any conversion project, the ease of the conversion is influenced by the way the initial project was implemented, such as using the register names and bit names from the data sheet (supplied in the Microchip include file). This along with other good programming techniques (symbolic code, documentation, ...) do a lot to ease the effort in a conversion project.

MODULE DIFFERENCES

First, one needs to understand what are the differences between the modules. Then the code can be evaluated to see if there are any changes required due to these differences. Some modules are functionally compatible and should only require minor changes due to the differences of the program and data memory maps of the devices. Other modules have differences due to the decision to keep module compatibility with the PICmicro Mid-Range family.

The following PIC17CXXX modules are upward compatible to the PIC18CXXX module. This means that the status and control bits are in the same registers at the same bit position. The PIC18CXXX module may have some additional control bits for the added features, but as long as the PIC17CXXX unimplemented bits were written as'0', the modules will operate in the same modes. PIC17CXXX modules that should not require source code modification to function on the PIC18CXXX family include:

- MSSP
- USART
- Hardware 8 x 8 Multiply

PIC17CXXX modules that will require some source code modification to function on the PIC18CXXX family include:

- 10-Bit A/D
- Timer0

PIC17CXXX modules that will require extensive source code modification to function on the PIC18CXXX family include:

- Timer 1
- Timer 2
- Timer 3
- Capture
- PWM
- In-Circuit Serial Programming (ICSP™)

A/D module

The PIC18CXXX 10-bit A/D module was specified to be compatible with the PIC16CXXX 10-bit A/D module. This means that there are some differences in the location of the status/control bits in the ADCON0 and ADCON1 registers. Table 1 shows which A/D control registers the bits reside in, and the comments indicate if the bit position changed or if it is in a different register.

Migration Impact

Code written for the PIC17C7XX 10-bit A/D module will require changes due to the remapping of the bit locations, as well as the differences for the program and data memory maps of the devices. The functionality of the module did not change, so the timing requirements should not need any modifications.

Note:	Please refer to the device data sheet for timing specifications to ensure applicability.

CCP Special Event Trigger

The CCP Special Event Trigger allows the compare action to start an A/D conversion. This feature is not present on the PIC17C7XX family and is an enhancement that does not affect code migration to the PIC18CXXX family.

TABLE 1: 10-BIT A/D BIT COMPATIBILITY

Bit	PIC17CXXX Register	PIC18CXX2 Register	Comments
ADON	ADCON0	ADCON0	— (1)
GO/$\overline{\text{DONE}}$	ADCON0	ADCON0	— (1)
CHS3	ADCON0	N.A.	PIC18CXX2 has up to 8 analog input channels. PIC17C7XX has up to 16 analog input channels.
CHS2	ADCON0	ADCON0	New bit position
CHS1	ADCON0	ADCON0	New bit position
CHS0	ADCON0	ADCON0	New bit position
ADCS2	N.A.	ADCON1	PIC18CXX2 has 3 new A/D Conversion Clock selections: Fosc/2, Fosc/4, and Fosc/16
ADCS1	ADCON1	ADCON0	Moved to different register
ADCS0	ADCON1	ADCON0	Moved to different register
ADFM	ADCON1	ADCON1	New bit position
PCFG3	ADCON1	ADCON1	— (1)
PCFG2	ADCON1	ADCON1	— (1)
PCFG1	ADCON1	ADCON1	— (1)
PCFG0	ADCON1	ADCON1	— (1)

Note 1: No change required

USART module

The PIC17CXXX has a USART module, while the PIC18CXXX has an Addressable USART (AUSART) module. The AUSART module is based on the PIC16CXXX family AUSART module, which has the high baud rate feature. All bits for the PIC17CXXX USART module have the same register names and the same bit position as the PIC18CXXX AUSART module. The AUSART module has two additional bits, the High Baud Rate Select (BRGH) bit and the Address Detect Enable (ADDEN) bit.

Table 2 shows the Addressable USART Register compatibility.

Migration Impact

Code written for the PIC17CXXX USART module should only require changes due to the differences for the program and data memory maps of the devices, but not due to the functionality of the module. The default state of the BRGH and ADDEN bits after a Power-on Reset allows compatibility with the PIC17CXXX USART module. Ensure that your code did not modify the state of these bits from the default state of '0'.

TABLE 2: ADDRESSABLE USART COMPATIBILITY

Bit	PIC17CXXX Register	PIC18CXX2 Register	Comments
CSRC	TXSTA	TXSTA	— (1)
TX9	TXSTA	TXSTA	— (1)
TXEN	TXSTA	TXSTA	— (1)
SYNC	TXSTA	TXSTA	— (1)
BRGH	N.A.	TXSTA	New bit
TRMT	TXSTA	TXSTA	— (1)
TX9D	TXSTA	TXSTA	— (1)
SPEN	RCSTA	RCSTA	— (1)
RX9	RCSTA	RCSTA	— (1)
SREN	RCSTA	RCSTA	— (1)
CREN	RCSTA	RCSTA	— (1)
ADDEN	N.A.	RCSTA	New bit
FERR	RCSTA	RCSTA	— (1)
OERR	RCSTA	RCSTA	— (1)
RX9D	RCSTA	RCSTA	— (1)

Note 1: No change required

Timer0 module

This module was specified to allow an operational compatibility to both the PIC16CXXX and PIC17CXXX families. Compatibility is specified by some new control bits. Table 3 shows the Timer0 Register compatibility. Figure 1 shows the PIC17CXXX Timer0 Block Diagram, while Figure 2 shows PIC18CXXX Timer0 Block Diagram when in 16-bit timer mode (T08BIT is cleared).

In the PIC17CXXX, the Timer0 module has the unique characteristic of having its own interrupt vector address. In PIC18CXXX devices, the Timer0 interrupt is included with all the other peripheral interrupts. Code conversions will need to take this into account.

Migration Impact

In the PIC18CXXX, the T08BIT selects if the Timer0 module will operate as an 8-bit timer or a 16-bit timer. To make this compatible with the PIC17CXXX implementation, the T08BIT must be cleared by software to select the 16-bit timer mode (the default state is set). When in the 16-bit timer mode, the 16-bit reads are now buffered. The TMR0H register is a buffered register that is loaded/written with an access to the TMR0L register. This allows removal of any software routines that were used to ensure a proper 16-bit read.

The PIC18CXXX PreScaler Assignment (PSA) bit selects if the prescaler is to be used. The default is prescaler not used, giving the same default prescale assignment as the PIC17CXXX Timer0. If the PSA bit is cleared, the prescaler is used. With the default state of the T0PS2:T0PS0 bits, the prescale assignment is 1:256. To assign this value to the PIC17CXXX, T0PS3 would need to be set and the T0PS2:T0PS0 bits would be don't care. For the PIC18CXXX, when the prescaler is selected all T0PS2:T0PS0 bits have meaning.

The PIC17CXXX Timer0 Interrupt vector address is no longer a dedicated location in the PIC18CXXX. The interrupt service routine is now required to test the TMR0IF bit as a potential interrupt source. Interrupt latency can be addressed by partitioning the interrupt sources between the high and low priority interrupt vector addresses. This technique is application dependent.

TABLE 3: TIMER0 REGISTER COMPATIBILITY

Bit	PIC17CXXX Register	PIC18CXX2 Register	Comments
TMR0ON	N.A.	T0CON	New bit to start Timer0 incrementing
T08BIT	N.A.	T0CON	New bit to configure timer in 16-bit mode
T0CS	T0STA	T0CON	New register
T0SE	T0STA	T0CON	New register and bit position
PSA	N.A.	T0CON	New register and bit position
T0PS3	T0STA	N.A.	— (1)
T0PS2	T0STA	T0CON	New register and bit position
T0PS1	T0STA	T0CON	New register and bit position
T0PS0	T0STA	T0CON	New register and bit position
INTEDG	T0STA	N.A.	— (1)

Note 1: This bit name is not applicable to the PIC18CXXX family.

FIGURE 1: PIC17CXXX TIMER0 MODULE BLOCK DIAGRAM

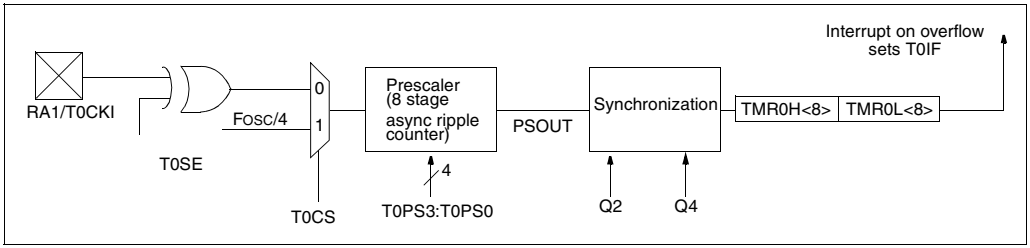
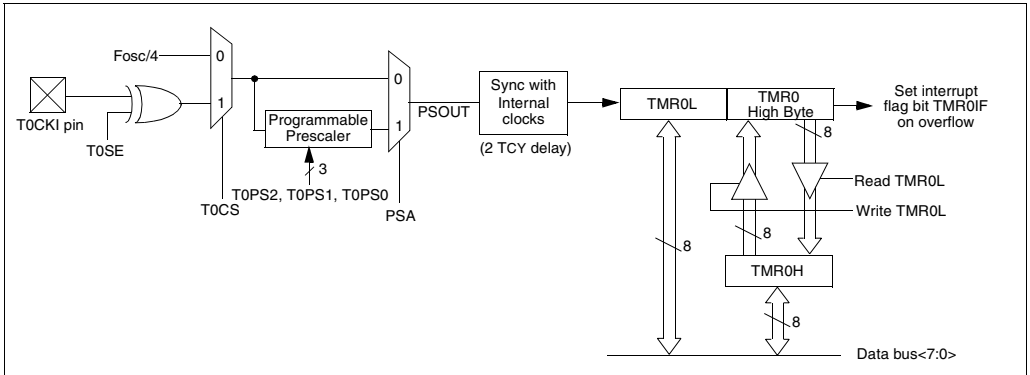


FIGURE 2: PIC18CXXX TIMER0 MODULE BLOCK DIAGRAM



Note 1: Upon reset, Timer0 is enabled in 8-bit mode with clock input from the T0CKI pin and the prescaler not selected (though prescaler bits select the maximum prescale count (1:256).

Timer1 module

The implementation of the PIC17CXXX Timer1 module is completely different from that on the PIC18CXXX. The module used on the PIC18CXXX family is the same implementation as the Timer1 module found on the PIC16CXXX with some enhancements. This module now allows a true implementation of a Real Time Clock circuit.

Figure 3 shows the Timer1 block diagrams for the PIC17CXXX. Operation in both the 8-bit and 16-bit modes are shown.

Figure 4 shows the Timer1 block diagram for the PIC18CXXX.

Migration Impact

This module requires a source code rewrite.

FIGURE 3: PIC17CXXX TIMER1 BLOCK DIAGRAMS

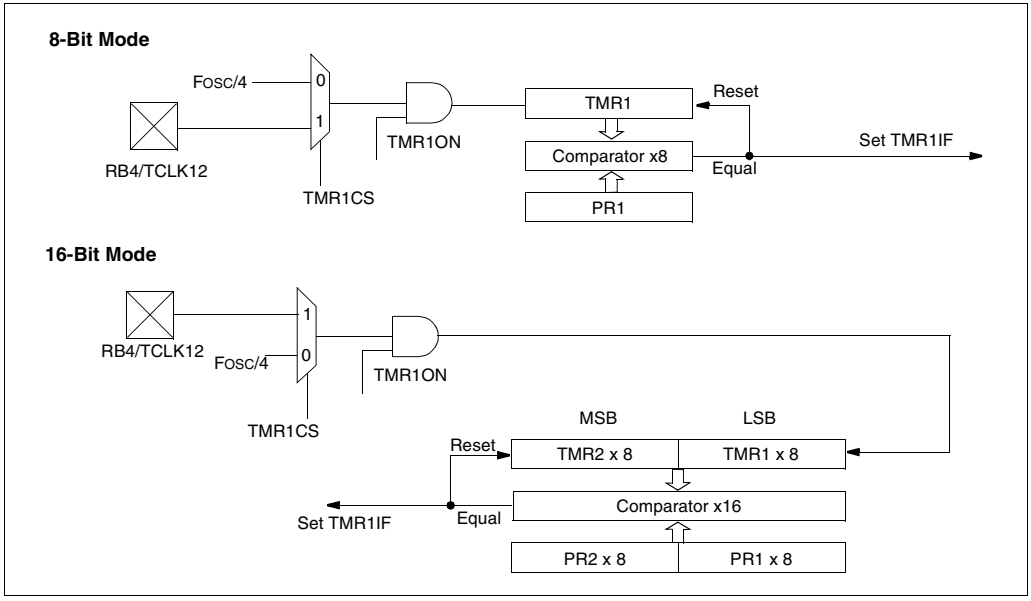
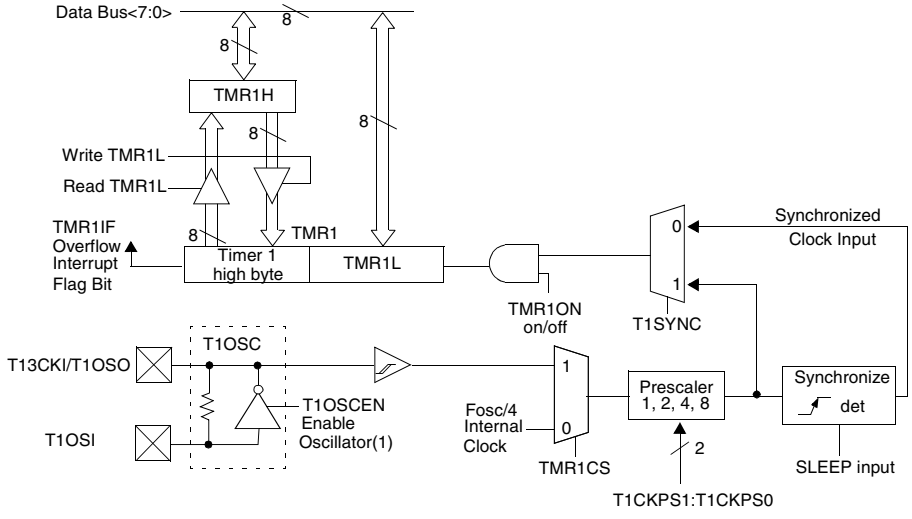


FIGURE 4: PIC18CXXX TIMER1 MODULE BLOCK DIAGRAM



Note 1: When enable bit T1OSCEN is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

Timer2 module

The implementation of the PIC17CXXX Timer2 module is completely different from that on the PIC18CXXX. The module used on the PIC18CXXX family is the same implementation as the Timer2 module found on the PIC16CXXX.

Figure 5 shows the Timer2 block diagrams for the PIC17CXXX. Operation in both the 8-bit and 16-bit modes are shown.

Figure 6 shows the Timer2 block diagram for the PIC18CXXX.

Migration Impact

This module requires a source code rewrite.

FIGURE 5: PIC17CXXX TIMER2 BLOCK DIAGRAMS

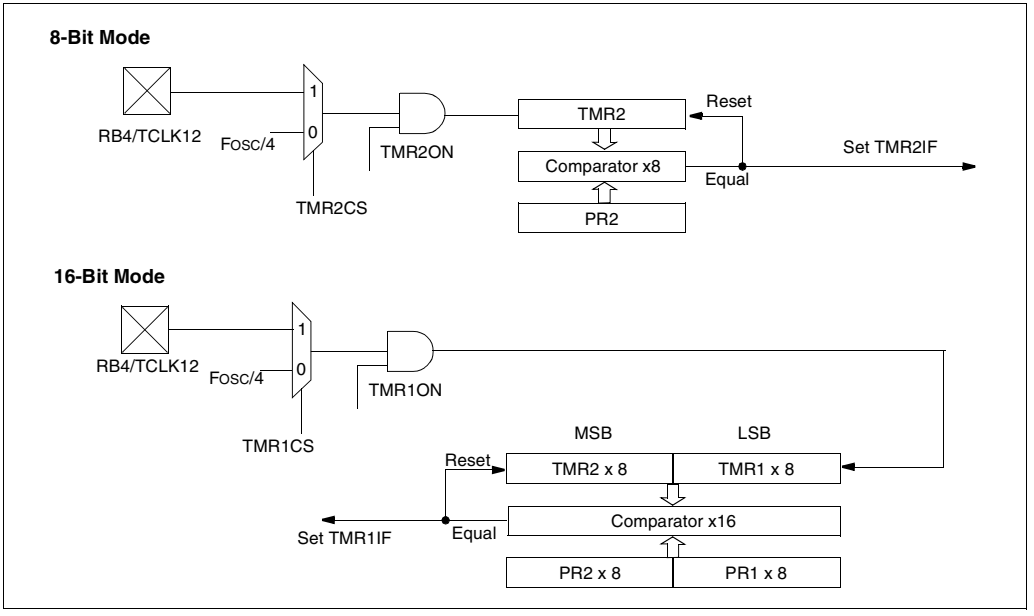
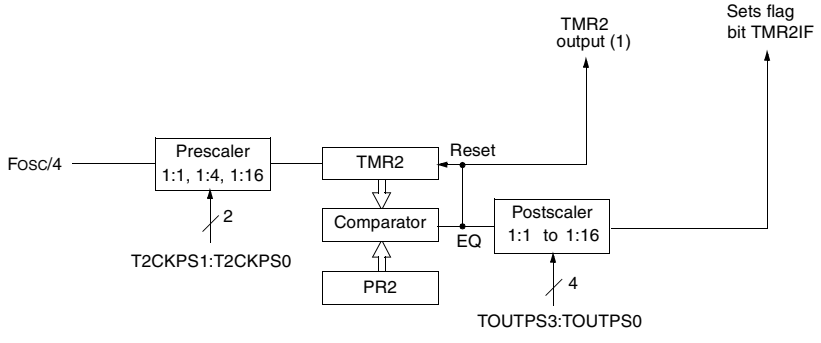


FIGURE 6: PIC18CXXX TIMER2 MODULE BLOCK DIAGRAM



Note 1: TMR2 register output can be software selected by the MSSP Module as a baud clock.

Timer3 module

The implementation of the PIC17CXXX Timer3 module is completely different from that on the PIC18CXXX. The module used on the PIC18CXXX family is the same implementation as the Timer1 module found on the PIC16CXXX, with some enhancements. This module now allows a true implementation of a Real Time Clock circuit.

Figure 7 is the block diagram of the PIC17CXXX Timer3 with three capture registers and one period register. Figure 8 is the block diagram of the PIC17CXXX Timer3 with four capture registers. As can be seen from these diagrams, the Timer3 module is tightly linked with the capture feature of the PIC17CXXX. In the PIC18CXXX, the capture feature is a software programmable mode of the CCP module.

Figure 9 is a block diagram of the PIC18CXXX Timer3 module.

Migration Impact

This module requires a source code rewrite.

FIGURE 7: PIC17CXXX TIMER3 WITH THREE CAPTURE AND ONE PERIOD REGISTER BLOCK DIAGRAM

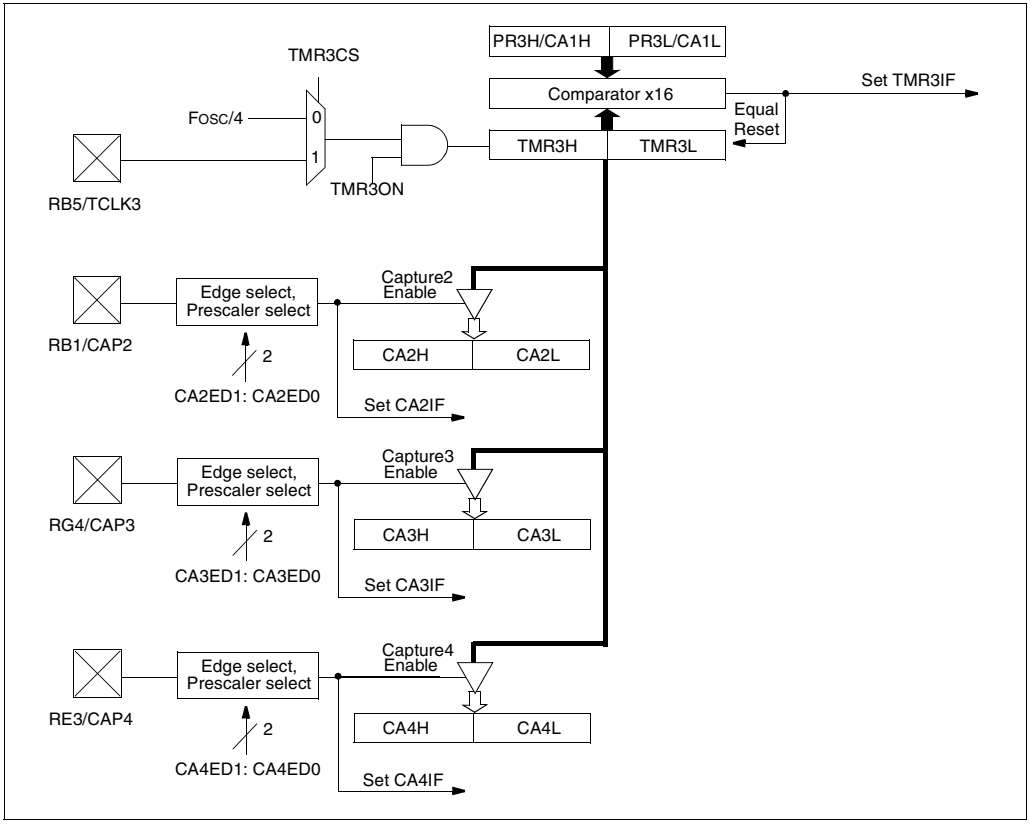


FIGURE 8: PIC17CXXX TIMER3 WITH FOUR CAPTURES BLOCK DIAGRAM

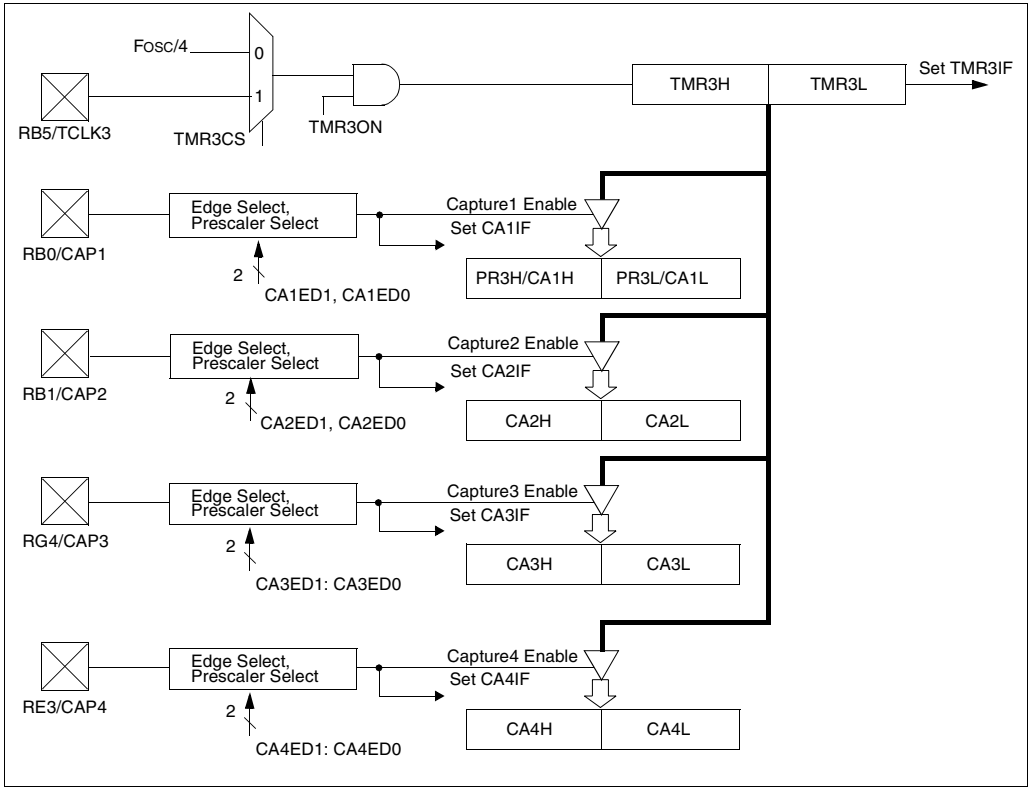
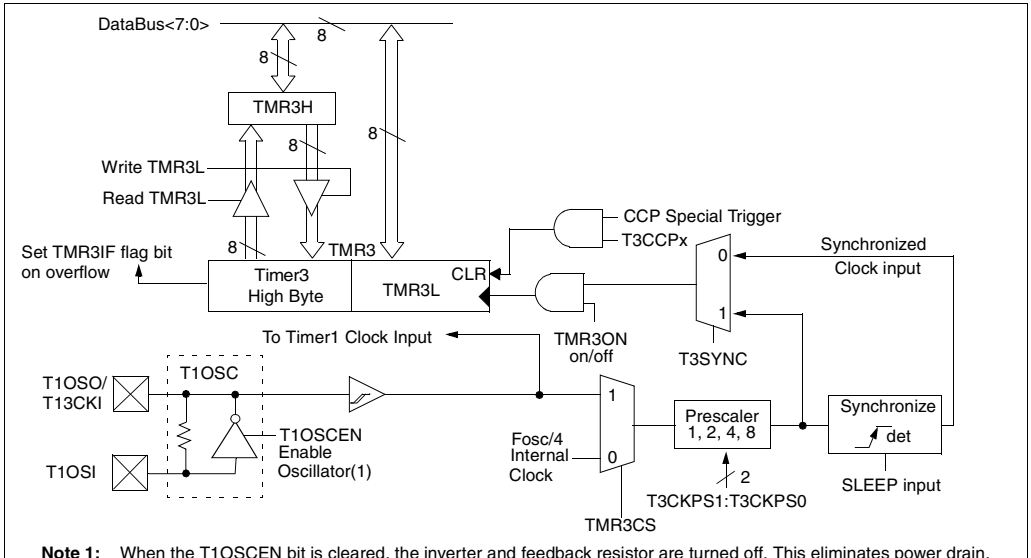


FIGURE 9: PIC18CXXX TIMER3 MODULE BLOCK DIAGRAM



Note 1: When the T1OSCEN bit is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

Capture/Compare/PWM modules

The PIC18CXXX family uses CCP modules. This is compatible with PIC16CXXX family devices. The PIC17CXXX allows more features to be used concurrently (3 PWM outputs and 4 capture inputs), while the PIC18CXX2 devices have 2 CCP modules. Table 4 shows the timer resources that are usable for the Time Based Operation feature selected.

TABLE 4: TIMER RESOURCES FOR TIME BASED OPERATION FEATURES

Time Based Feature	PIC17CXXX	PIC18CXXX
Capture	Timer3	Timer1 or Timer3
Compare	N.A.	Timer1 or Timer3
PWM	Timer1 or Timer2	Timer2

PWM Operation

In the PIC17CXXX, the PWM time base can be set to either Timer1 or Timer2. These timers both have the capability to have their clock source derived from the external pin TCLK12. The PIC18CXXX PWM must always use Timer2 as the time base with the clock source from the internal device clock. Table 5 shows the registers used to specify the PWM duty cycle between the two families.

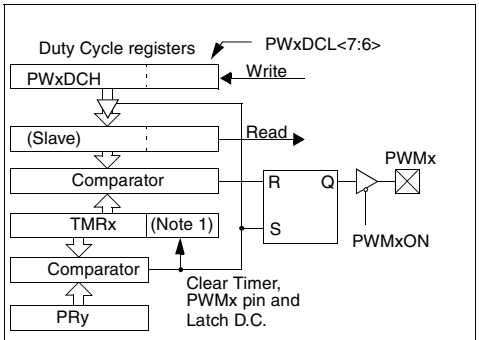
Figure 10 is a block diagram of the PIC17CXXX PWM. Timer1 or Timer2 may be used as the time base for the PWM outputs.

Figure 11 is a block diagram of the PIC18CXXX PWM. Timer2 is the time base for all PWM outputs.

TABLE 5: DUTY CYCLE REGISTERS

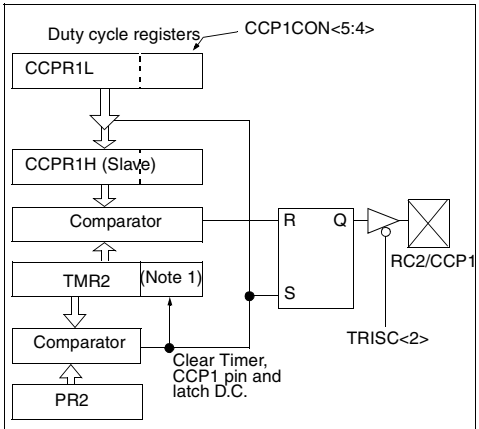
Device	PWM Duty Cycle Bits	
	DC9:DC2	DC1:DC0
PIC17CXXX	PWxDCH	PWxDCL<7:6>
PIC18CXXX	CCPRxL	CCPxCON<5:4>

FIGURE 10: PIC17CXXX PWM BLOCK DIAGRAM



Note 1: 8-bit timer is concatenated with 2-bit internal Q clock or 2 bits of the prescaler to create 10-bit time-base.

FIGURE 11: PIC18CXXX PWM BLOCK DIAGRAM



Note: 8-bit timer is concatenated with 2-bit internal Q clock or 2 bits of the prescaler to create 10-bit timebase.

MIGRATION IMPACT

Migrating code from the PIC17CXXX family to the PIC18CXXX family will require a rewrite of the source code to function. Since the CCP module is software programmable to operate in any of the three modes, the total number of PWM outputs may not match what is provided by the PIC17CXXX.

Capture Operation

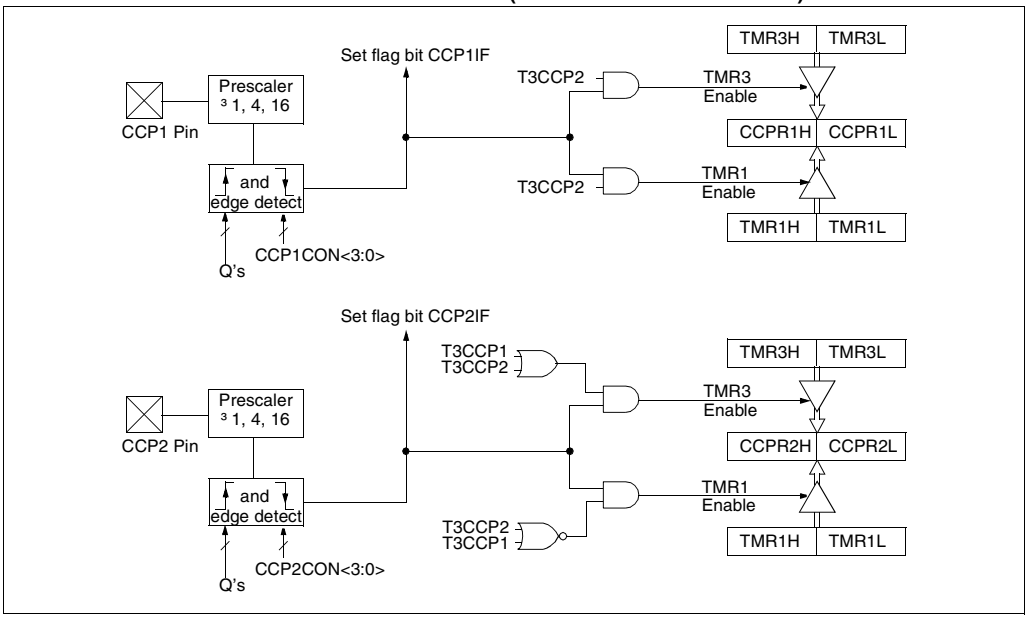
In the PIC17CXXX family, the capture feature is tightly linked with the Timer3 module. Figure 7 and Figure 8 show the capture block diagrams.

Figure 12 is the PIC18CXXX Capture Operation Block Diagram. In the PIC18CXXX, the capture feature is a software programmable mode of the CCP module.

MIGRATION IMPACT

Migrating code from the PIC17CXXX family to the PIC18CXXX family will require a rewrite of the source code to function. Since the CCP module is software programmable to operate in any of the three modes, the total number of capture inputs may not match what is provided by the PIC17CXXX.

FIGURE 12: PIC18CXXX CAPTURE OPERATION (WITH TIMER1 AND TIMER3)



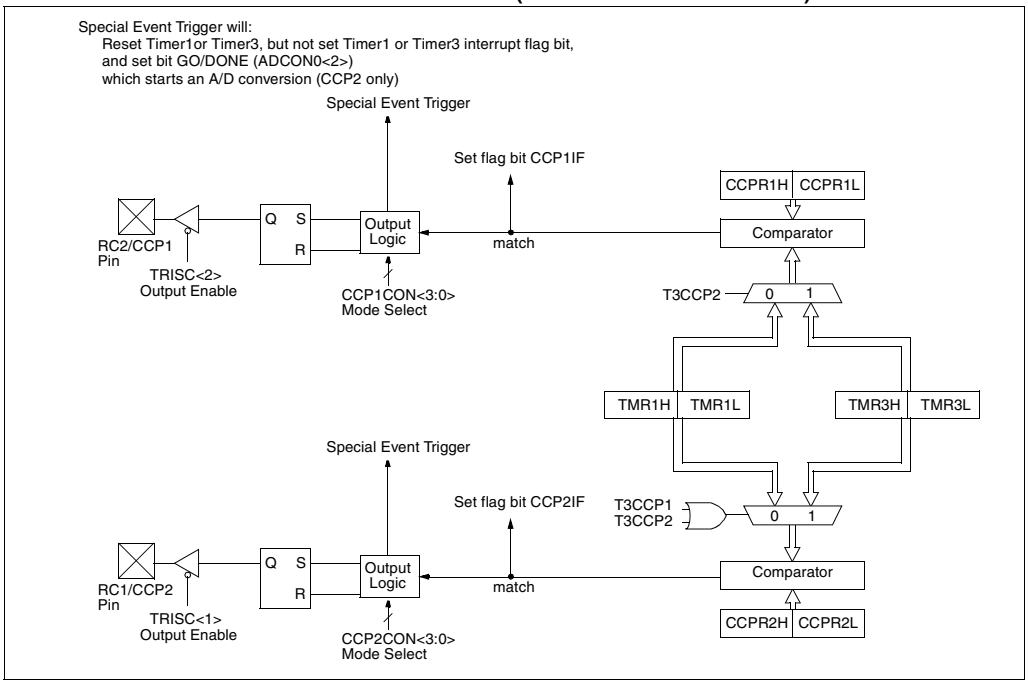
Compare Operation

The PIC17CXXX family does not support a compare operation. This is an enhancement for the PIC18CXXX family. Figure 13 shows the operation of the PIC18CXXX compare mode. Two compare values can be initialized, and they can be used to compare against either Timer1 or Timer3.

MIGRATION IMPACT

This feature of the CCP module is an enhancement to the PIC17CXXX devices.

FIGURE 13: PIC18CXXX COMPARE OPERATION (WITH TIMER1 AND TIMER3)



Master SSP Module

The PIC17CXXX MSSP module is upwardly compatible with the PIC18CXXX MSSP module. The PIC18CXXX MSSP module also includes two modes that are present in the PIC16CXXX SSP module. These are the modes:

1. I^2C slave mode, 7-bit address with start and stop bit interrupts enabled
2. I^2C slave mode, 10-bit address with start and stop bit interrupts enabled

These modes were retained for ease of code migration from PIC16CXXX devices to the PIC18CXXX family.

Migration Impact

Code written for the PIC17CXXX MSSP module should only require changes due to the differences in the program and data memory maps of the devices, but not due to the functionality of the module.

External Interrupts

For the PIC17CXXX, the INT interrupt had its own vector address. In the PIC18CXXX, it is part of the peripheral interrupts vector address. This means that the INT interrupt code will need to be moved into the general peripheral interrupt service routine (ISR), and this routine will need to add a check for the INT interrupt source.

The PIC18CXXX family has some enhancements for the external interrupts. First, there are now three external interrupt pins, as opposed to one pin in the PIC17CXXX family. Second, enhancements to the architecture of the interrupt logic allows additional capability (High/Low priority). These enhancements are discussed in the section "Architectural Enhancements" .

Migration Impact

The PIC17CXXX external interrupt requires minor modifications to be used with the PIC18CXXX devices.

PortB Interrupt-On-Change

The PORTB interrupt-on-change feature of the PIC17CXXX family has all PORTB pins with the interrupt on change feature. This feature was multiplexed with other peripheral features such as Captures, PWMs, Timer clock inputs, and SPI pins. The PORTB interrupt on change feature of the PIC18CXXX family matches that of our Mid-Range family. That is, there is only an interrupt on change on the upper four port pins of PORTB. There are no other peripheral feature multiplexed onto these pins.

Migration Impact

On the PIC18CXXX family, only RB7:RB4 have the interrupt on change feature. These pins do not have any peripheral feature multiplexed on them.

PORTB Weak Pull-up Enable

The control bit to enable the weak pull-ups on PORTB have been moved from PORTB<7> (PIC17CXXX) to INTCON2<7> (PIC18CXXX).

Migration Impact

Code changes are only required due to the differences of the data memory maps.

Hardware 8 x 8 Multiply

The operation of the 8 x 8 hardware multiply is identical between the two families.

Migration Impact

Changes may only be required due to the differences of the data memory maps.

Brown-out Reset (BOR)

The Brown-out Reset (BOR) logic has been enhanced in the PIC18CXXX family. The BOR trip point is now programmable at time of device programming. One of four trip points can be selected. The BOR trip points are shown in Table 6.

Migration Impact

Since none of these trip points are specified at the same voltage level as the trip point for the PIC17CXXX family, some modifications may need to be done with the application. These modifications may be software, hardware, or both.

TABLE 6: BOR TRIP POINT COMPARISON

Family	BOR Trip Point Option				
	4.5 V (min)	4.2 V (min)	4.0 V (typ)	2.7 V (min)	2.5 V (min)
PIC18CXXX	Yes	Yes	—	Yes	Yes
PIC17CXXX	—	—	Yes	—	—

On-Chip Oscillator Circuit

The oscillator circuit has been modified to allow new enhanced features, such as a Phase Lock Loop (PLL) option) and clock switching to the Timer1 oscillator. Clock switching allows the optimization of the applications power consumption, by only operating at high frequency (high power) when the application software requires that performance. It also allows the operation at a lower frequency (low power) when application software is not performance critical.

The oscillator options of the PIC18CXXX family allow an extended frequency range compared to the PIC17CXXX device. The oscillator mode that was selected for the PIC17CXXX device may need to be changed to operate the PIC18CXXX at the desired frequency.

Table 7 shows a comparison of the oscillator selection modes between the PIC17CXXX and the PIC18CXXX devices. There are some modes where an additional I/O pin becomes available to the device. These are the RCIO and ECIO modes.

Migration Impact

Since the oscillator circuitry is different between the two families, any external components that are required need to be re-evaluated to ensure operation in the application.

Note:	Oscillator operation should be verified to ensure that it starts and performs as expected. Adjusting the loading capacitor values and/or the oscillator mode may be required.

$\overline{\text{MCLR}}$

The $\overline{\text{MCLR}}$ operation is different between the two families. The $\overline{\text{MCLR}}$ operation of the PIC18CXXX family is identical to the Mid-Range family. Please inspect electrical specification parameter # 30 to understand the implications in your system.

Migration Impact

Ensure that the differences in the electrical specifications are met by the application circuit.

Power-On Reset (POR)

The Power-On Reset (POR) operation is different between the two families. The POR operation of the PIC18CXXX family is identical to the Mid-Range family (for the same modes).

Migration Impact

Ensure that the differences in the Power-On Reset timings are addressed by the hardware and software of the application.

In-Circuit Serial Programming (ICSP)

The ICSP operation is different between the two families. This relates to both the hardware interface as well as the software protocol and timings.

Migration Impact

The new implementation method will need to be accounted for in the design conversion.

TABLE 7: OSCILLATOR MODE SELECTION COMPARISON

Frequency Range	Oscillator Type	Oscillator Mode Selection		Comment
		PIC17CXXX	PIC18CXXX	
DC - 4 MHz	RC	RC	RC or RCIO	—
DC - 200 kHz	Crystal/Resonator	LF	LP	—
200 KHz - 2 MHz	Crystal/Resonator	LF	XT	—
2 MHz - 4 MHz	Crystal/Resonator	XT	XT	—
4 MHz - 16 MHz	Crystal/Resonator	XT	HS	—
16 MHz - 25 MHz	Crystal/Resonator	XT	HS or HS + PLL (1)	—
25 MHz - 33 MHz	Crystal/Resonator	XT	HS + PLL (2)	—
33 MHz - 40 MHz	Crystal/Resonator	N.A.	HS + PLL (3)	—
DC - 33 MHz	External Clock	EC	EC or ECIO	—
33 - 40 MHz	External Clock	N.A.	EC or ECIO	—

Note 1: The external crystal would have a frequency of 4 MHz - 6.25 MHz.

Note 2: The external crystal would have a frequency of 6.25 MHz - 8.25 MHz.

Note 3: The external crystal would have a frequency of 8.25 MHz - 10 MHz.

MEMORY MAP DIFFERENCES

The memory map affects instructions that are required for program flow and addressing program and data memory. The memory maps between the PIC17CXXX and PIC18CXXX families are similar, but still require discussion for the upward migration of application code.

These are broken down into two discussions, one for the Program Memory map and the other for the Data Memory map.

Program Memory

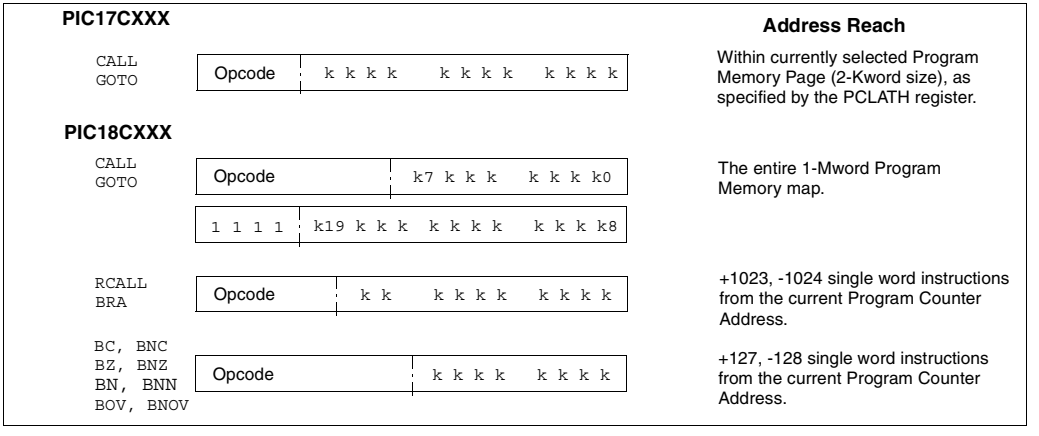
The PIC17CXXX family can address 64-Kwords of program memory (128-KBytes). This memory space is broken up into 8 program memory pages of 8-Kwords. The architecture required the modification of the PCLATH register for any CALL or GOTO instruction that has a destination in a different page than is currently selected by the PCLATH register.

The PIC18CXXX family can address 2-MBytes of program memory (1-Mword). The use of program memory pages has been eliminated. Now the CALL and GOTO instructions are 2-word instructions and can address any location in the program memory space. In some instances the destination address is close to the CALL or GOTO instruction. In these cases, optimized instructions are available; the relative call and unconditional branch instructions (called the RCALL and BRA instructions), which are one word instructions. Condition branch instructions are also available, which will branch to a new program memory location based on an offset from the current program counter value. These conditional branch instructions are useful for the generation of optimized code from a C compiler. Figure 14 shows the program flow instructions for PIC17CXXX and PIC18CXXX families.

Example 1 shows a code sequence for branching to a code segment depending on the status of the zero bit. For the PIC17CXXX family, the GOTO instruction will cause the execution to branch to the program memory page dependent on the value loaded in the PCLATCH register. In the PIC18CXXX family, the GOTO instruction is two words and can address any location in the program memory. To ensure robustness of the system, the 2nd word of a two word instruction (when executed as an instruction) is executed as a NOP. This allows the same source code to work for both families, though in the PIC18CXXX family an extra instruction cycle will be required to reach the code at the Not_Zero symbol.

Example 2 shows an alternate implementation done with the PIC18CXXX instruction set. With this instruction, the location of the software routine labeled Zero would need to be within 128 words before the BZ instruction or 127 words after the BZ instruction.

FIGURE 14: PROGRAM MEMORY FLOW INSTRUCTIONS



EXAMPLE 1: PIC17CXXX OR PIC18CXXX CODE EXAMPLE

```
          BTFSC STATUS, Z      ; Is result Zero
          GOTO  Zero           ; YES, goto the code for a result of Zero
Not_Zero  :                    ; NO, result was not Zero
```

EXAMPLE 2: ALTERNATE PIC18CXXX CODE EXAMPLE

```
          BZ    Zero           ; If result Zero, goto the code for a result of Zero
 Not_Zero :                    ; NO, result was not Zero
```

The ability to branch on the condition of a status bit value allows more efficient code to be generated. Table 8 shows how these are implemented between the PIC18CXXX family and the PIC17CXXX family. In the PIC18CXXX family, the branch is relative from the program counter location and has a reach of -128 words or +127 words. In the PIC17CXXX family two possible methods are shown. Method 1 is the positive logic method which may have been used. Method 2 (shaded) is the negative logic method which would get to the carry routine one instruction cycle quicker. Method 1 is what translates to the corresponding PIC18CXXX instruction.

Each method requires the use of a GOTO instruction. The GOTO instruction allows access to any location in the selected page of program memory (as specified by the value in the PCLATH register). The number of cycles indicates the number of cycles to get to the desired routine for the true case (as defined by the PIC18CXXX conditional branch instruction) and the cycles in parentheses () indicates the number of cycles for the false case. As can be seen by the Table 8 comparison, the number of cycles and memory requirements is better for the PIC18CXXX instructions.

Migration Impact

Minimal changes should be required for PIC17CXXX source code. Any operations on PCLATH (paging) are ignored, since the PIC18CXXX families CALL and GOTO instructions contain the entire address.

Look-up tables will require some sort of modification. Tables implemented using the RETLW instruction need to be modified due to the Program Counter now being a byte counter (see explanation in "Program Counter"). Tables implemented using the PIC17CXXX Table Reads will need to be modified to address the new implementation in the PIC18CXXX (see explanation in "Table Reads and Table Writes").

Code optimization can be achieved by removing instructions that modify the PCLATH register before the CALL and GOTO instructions. If the desired program memory location is within ± 1K instruction words, then the use of the BRA and RCALL instructions will maintain the use of one instruction word, instead of the new requirement for two words.

Additional optimization can be achieved by utilizing new instructions, such as Branch on condition instructions. These Branch on condition instructions must have the program memory address of the branch code to be within -128 to +127 instruction words from the branch instruction.

TABLE 8: BRANCH ON STATUS BIT COMPARISON

Alternate PIC18CXXX Instruction			PIC17CXXX Instruction Sequence (Note 1)								
Method		Cycles/ Words	Method 1				Cycle/ Words	Method 2			Cycle/ Words
BC	Carry	2 (1) /1		BTFSC	STATUS, C		3 (2) /2		BTFSS	STATUS, C	2 (3) /2
				GOTO	Carry				GOTO	NoCarry	
			NoCarry	:				Carry	:		
BNC	NoCarry	2 (1) /1		BTFSS	STATUS, C		3 (2) /2		BTFSC	STATUS, C	2 (3) /2
				GOTO	NoCarry				GOTO	Carry	
			Carry	:				NoCarry	:		
BN	Neg	2 (1) /1		BTFSC	STATUS, N		3 (2) /2		BTFSS	STATUS, N	2 (3) /2
				GOTO	Neg				GOTO	NotNeg	
			NotNeg	:				Neg	:		
BNN	NotNeg	2 (1) /1		BTFSS	STATUS, N		3 (2) /2		BTFSC	STATUS, N	2 (3) /2
				GOTO	NotNeg				GOTO	Neg	
			Neg	:				NotNeg	:		
BOV	Ovflw	2 (1) /1		BTFSC	STATUS, OV		3 (2) /2		BTFSS	STATUS, OV	2 (3) /2
				GOTO	Ovflw				GOTO	NoOvflw	
			NoOvflw	:				Ovflw	:		
BNOV	NoOvflw	2 (1) /1		BTFSS	STATUS, OV		3 (2) /2		BTFSC	STATUS, OV	2 (3) /2
				GOTO	NoOvflw				GOTO	Ovflw	
			Ovflw	:				NoOvflw	:		
BZ	Zero	2 (1) /1		BTFSC	STATUS, Z		3 (2) /2		BTFSS	STATUS, Z	2 (3) /2
				GOTO	Zero				GOTO	NotZero	
			NotZero	:				Zero	:		
BNZ	NotZero	2 (1) /1		BTFSS	STATUS, Z		3 (2) /2		BTFSC	STATUS, Z	2 (3) /2
				GOTO	NotZero				GOTO	Zero	
			Zero	:				NotZero	:		

Note 1: This method may also be used by the PIC18CXXX family. This is source code compatible, but the GOTO instruction is now a two word instruction. When the second word of the GOTO instruction is executed as if it was a single word instruction (when the skip occurs), the second word is executed as a no operation (NOP instruction).

Data Memory

The Data Memory Map of the PIC17CXXX devices is shown in Figure 15 with the PIC18CXXX data memory map shown in Figure 16. In both architectures, the bank size is 256 bytes. Software code migration does not require the low nibble of the Bank Select Register (BSR) to be modified. The PIC17CXXX devices also bank the Special Function Registers (SFRs). This is not required with the PIC18CXXX architecture. All instructions which are used to modify the BSR<7:4> bit may be removed from the user code.

Figure 17 shows the mapping of data memory from a PIC17CXXX device to a PIC18CXXX device. The mapping translates without effort given that the GPR RAM addresses were specified with the full address, and not the relative address within the selected bank. With a full 10-bit address, the assembler will map the addresses correctly. The SFR, though not at the same addresses will be properly mapped to the correct location in bank 15 due to the supplied header file. No software coding modifications are required to address the SFR registers, since the SFR registers are in the Access bank, and can be addressed regardless of the selected bank (value of the BSR register).

The low nibble of the BSR (BSR<3:0>) specifies the RAM bank to access. This is the same for both the PIC17CXXX and PIC18CXXX. Any operations on the high nibble of the BSR (BSR<7:4>) can be ignored by the PIC18CXXX, since these bits are not implemented.

In the PIC17CXXX, the GPRs in the memory range 1Ah to 1Fh are in shared RAM. When mapped to the PIC18CXXX, these addresses are in the access bank and therefore are also shared RAM.

FIGURE 15: PIC17CXXX DATA MEMORY MAP

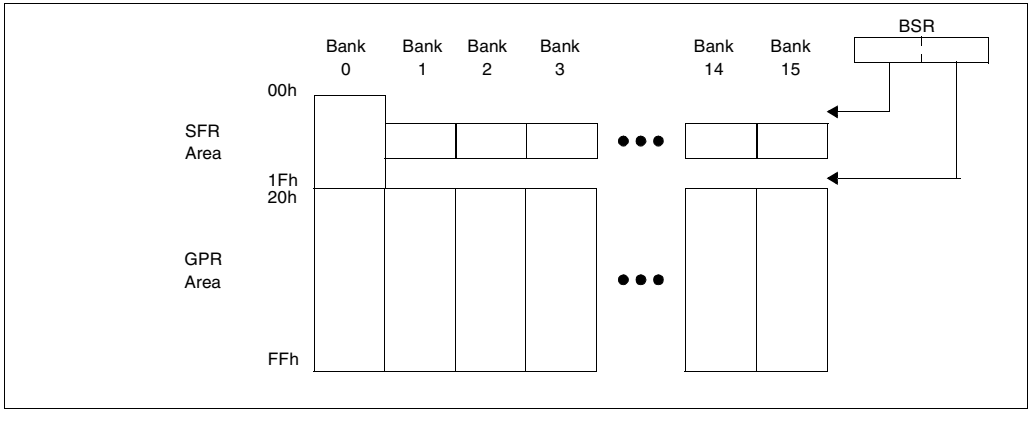


FIGURE 16: PIC18CXXX DATA MEMORY MAP

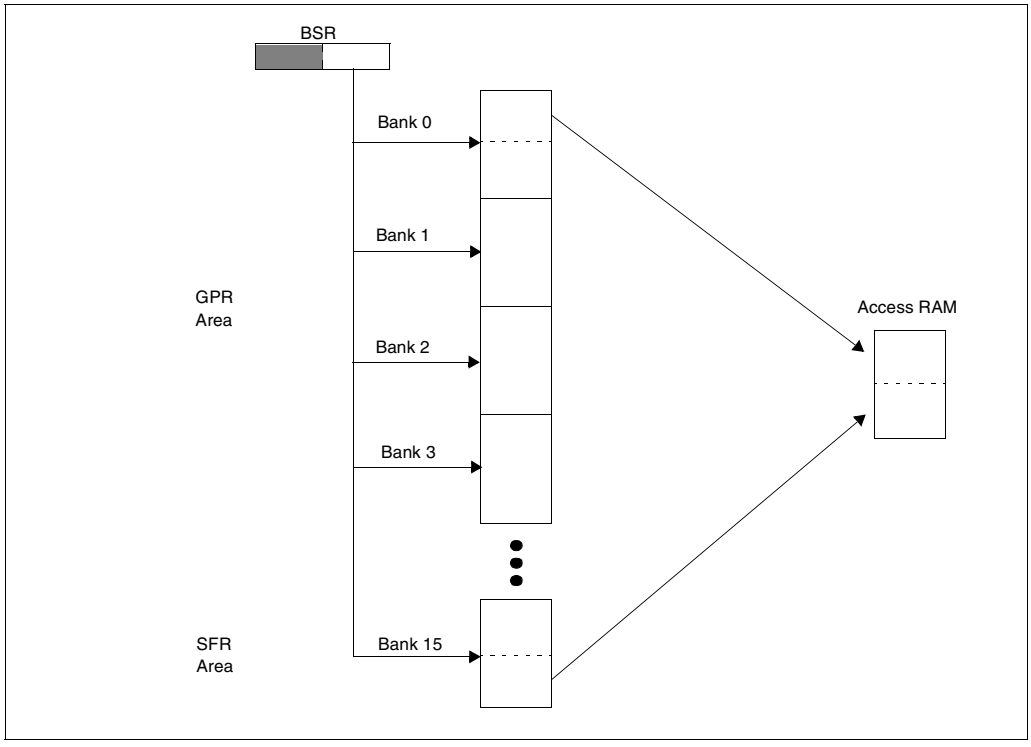
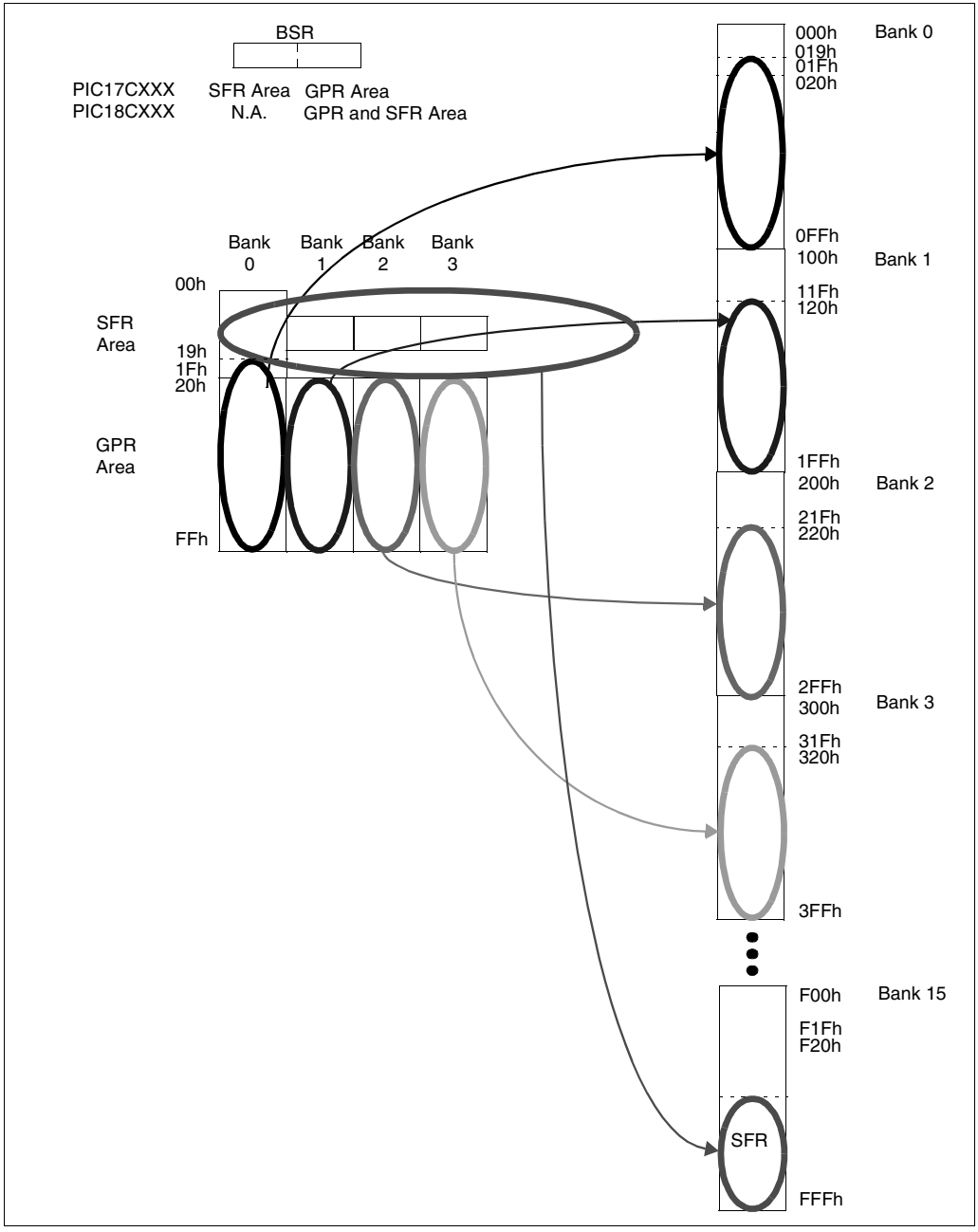


FIGURE 17: MAPPING OF DATA MEMORY FROM PIC17CXXX TO PIC18CXXX



There are occasions where the application software requires moving data from one register to another. This transfer may be a single byte or a block of data.

For the PIC17CXXX family there are instructions that move the contents from the Peripheral (P) area (first 32 locations in data memory) to the File (F) area (anywhere in the specified 256 locations), or from the File area to the Peripheral area. The actual RAM address is specified by the Bank Select Register (BSR) value, since both the SFR registers and GPR registers are banked.

With the PIC18CXXX, the instruction moves the contents from one register to another anywhere in the 4 KByte data memory space without any banking requirements.

Example 1 shows the sequence of instructions to move a value from one RAM location to another in the PIC17CXXX family. Example 2 shows the instruction to move a value from one RAM location to another in the PIC18CXXX family.

Migration Impact

If the PIC17CXXX source code uses register definitions that specifies full 12-bit addresses for ALL register file locations, then no work is required for the remapping of the data memory. The BSR low nibble (BSR<3:0>) will be updated in the same fashion. Optimization can be done by removing any instructions that are used to modify the high nibble of the BSR register (BSR<7:4>), since these bits are not implemented on the PIC18CXXX. There is one instruction that is only used to do this modification. This is the `MOVLR` instruction.

EXAMPLE 1: PIC17CXXX MEMORY-TO-MEMORY MOVES

Case 1: Single Byte Transfer

```
            banksel     MYREG1              ; Switch to the bank for MYREG1
                                            ;   (may not be required)
            MOVFP MYREG1, WREG              ; Move contents of MYREG1 to the
                                            ; WREG register
            banksel     MYREG2              ; Switch to the bank for MYREG2
                                            ;   (may not be required)
            MOVPF WREG, MYREG2              ; Move contents of the WREG
                                            ; register to MYREG2
```

Case 2: Block Transfer

```
            MOVLW BYTE_CNT                  ; Load the Byte Count value
            MOVWF CNTR                      ; into register CNTR (same bank as MYREG1)
LP1         banksel     MYREG1              ; Switch to the bank for MYREG1
                                            ;   (may not be required)
            MOVFP MYREG1, WREG              ; Move contents of MYREG1 to the
                                            ;   WREG register
            banksel     MYREG2              ; Switch to the bank for MYREG2
                                            ;   (may not be required)
            MOVPF WREG, MYREG2              ; Move contents of the WREG
                                            ;   register to MYREG2
            DECFSZ  CNTR                    ; All bytes moved?
            BRA     LP1                     ; NO, move next byte
Continue    :                               ; YES, Continue
```

EXAMPLE 2: PIC18CXXX MEMORY-TO-MEMORY MOVES

Case 1: Single Byte Transfer

```
            MOVFF MYREG1, MYREG2            ; Move contents of MYREG1 to MYREG2
```

Case 2: Block Transfer

```
            MOVLW BYTE_CNT                  ; Load the Byte Count value
            MOVWF CNTR                      ; into register CNTR
LP1         MOVFF POSTINC0, POSTINC1        ; Move contents of MYREG1 to MYREG2
            DECFSZ  CNTR                    ; All bytes moved?
            BRA     LP1                     ; NO, move next byte
Continue    :                               ; YES, Continue
```

INSTRUCTION SET

With the merging of the PIC16CXXX and PIC17CXXX instruction sets to create the PIC18CXXX instruction set and the enhancements to the architecture, some instructions had to be modified. Table 9 shows the PIC17CXXX instructions that have been modified. Some instructions operate on a new status bit which indicates if the resultant value is negative (N). Five instructions now affect the status of the zero (Z) bit. These instructions are:

- CLRF
- RRCF
- RRNCF
- RLCF
- RLNCF

Three instructions have changed the mnemonics, but the arguments do not need to be modified. For these three instructions a simple search and replace can be used. These instructions are:

- MOVPF
- MOVFP
- NEGW

Table 9 shows what these instructions should be replaced with.

The method of operation for four instructions (Table Reads and Table Writes) has changed. The application code surrounding the operation of this feature needs to be revisited and the code modified accordingly. These instructions are:

- TABLRD
- TLRD
- TABLWT
- TLWT

Lastly, two instructions have been removed. These instructions are:

- MOVLR
- LCALL

The MOVLR instruction is no longer required since there are no separate banks for the Special Function Registers. The LCALL instruction is changed to the PIC18CXXX CALL instruction, since it can access any location in the program memory map. The application code that preconditioned the PCLATH register can be removed, since it is no longer needed for calling the desired routine.

The PIC17CXXX family only has five instructions where the operation on the status bits changed. These are the clear file and rotate instructions (CLRF, RLCF, RLNCF, RRCF, and RRNCF).

Table 9 shows the PIC17CXXX instructions that are different in the PIC18CXXX architecture. These differences may be related to the status bits that are affected. An instruction is now handled by a more generic instruction, or the operation of the instruction has been modified to better fit with the new architecture. Rows that are shaded are new instructions to the PIC18CXXX architecture that are replacing PIC17CXXX instructions. These are shown to indicate the status bits affected.

Migration Impact

Ensure that the instructions that affect the status bits differently do not cause algorithm issues and that other instructions are appropriately converted and implemented.

TABLE 9: INSTRUCTION SET COMPARISON

Instruction	Status Bits Affected		Comment
	PIC17CXXX	PIC18CXXX	
ADDLW k	C,DC,OV, Z	C,DC,OV, Z, **N**	—
ADDWF f, d	C,DC,OV, Z	C,DC,OV, Z, **N**	—
ADDWFC f, d	C,DC,OV, Z	C,DC,OV, Z, **N**	—
ANDLW k	Z	Z, **N**	—
ANDWF f, d	Z	Z, **N**	—
CLRF f, s	none	**Z**	Instruction now affects Zero (Z) bit
DECF f, d	C,DC,OV, Z	C,DC,OV, Z, **N**	—
INCF f, d	C,DC,OV, Z	C,DC,OV, Z, **N**	—
IORLW k	Z	Z, **N**	—
IORWF f, d	Z	Z, **N**	—
LCALL k	none	N.A.	Use CALL n, s instruction
MOVFP f, p	none	N.A.	Use MOVFF fs, fd instruction
MOVFF fs, fd	N.A.	none	Replaced MOVFP and MOVPF instructions
MOVLR k	none	N.A.	Not required for PIC18CXXX devices
MOVPF p, f	Z	N.A.	Use MOVFF fs, fd instruction
NEGW f, s	C,DC,OV, Z	N.A.	Use NEGF f instruction
NEGF f	N.A.	C,DC,OV, Z, **N**	Replaced NEGW f, s instruction
RLCF f, d	C	C, **Z, N**	Instruction now affects Zero (Z) bit
RLNCF f, d	none	**Z, N**	Instruction now affects Zero (Z) bit
RRCF f, d	C	C, **Z, N**	Instruction now affects Zero (Z) bit
RRNCF f, d	none	**Z, N**	Instruction now affects Zero (Z) bit
SUBLW k	C,DC,OV, Z	C,DC,OV, Z, **N**	—
SUBWF f, d	C,DC,OV, Z	C,DC,OV, Z, **N**	—
SUBWFB f, d	C,DC,OV, Z	C,DC,OV, Z, **N**	—
TABLRD t, i, f	none	N.A.	Use TBLRD instructions
TABLWT t, i, f	none	N.A.	Use TBLWT instructions
TLRD t, f	none	N.A.	Use TBLRD instructions
TLWT t, f	none	N.A.	Use TBLWT instructions
TBLRD* TBLRD*+ TBLRD*- TBLRD+*	N.A.	none	Replaced TABLRD and TLRD instructions
TBLWT* TBLWT*+ TBLWT*- TBLWT+*	N.A.	none	Replaced TABLWT and TLWT instructions
XORLW k	Z	Z, **N**	—
XORWF f, d	Z	Z, **N**	—

Note 1: The N bit is new for the PIC18CXXX family of devices.

ARCHITECTURAL ENHANCEMENTS

Some of the architectural enhancements that are implemented in the PIC18CXXX family include:

- Program Counter
- Table Read / Table Write
- Interrupts
- Stack
- Indirect Addressing

Program Counter

The program counter of the PIC18CXXX Architecture works on a byte address, as opposed to a word address for the PIC17CXXX family. This means that the addresses of routines will be different. When using symbolic coding, the assembler will take care of generating the correct address, but any routine that directly modifies the program counter needs to take this difference into account. One of the most common code functions where this occurs is in table lookup routines that use the `RETLW` instruction.

Example 1 shows a typical table look-up for the PIC17CXXX family. Example 2 shows the table look-up for the PIC18CXXX Architecture. Since the Offset needs to be multiplied by two to get the byte address, the reach (size) of the look-up table is now half. Access to the PCLATU and PCLATH registers or the ability to do Table Reads allows larger tables to be stored in memory.

Occasionally the use of the '$' symbol is used in the source code to indicate the program address of the current instruction. The '$' syntax still operates as before, but since the program counter now specifies byte addresses any offset to the '$' parameter need to be doubled. Example 3 shows these modifications.

Migration Impact

Any modification of the PCL register will require the source code to be inspected to ensure that the desired address will be accessed. This is due to the Program Counter being a byte count into program memory and not the program memory word count. This is commonly found in simple Table Lookup routine. Remember that reading PCL updates the contents of PCLATH and PCLATU (from PCH and PCU), and writing to PCL loads PCH and PCU with the contents of PCLATH and PCLATU.

EXAMPLE 1: PIC17CXXX TABLE LOOK-UP USING THE `RETLW` INSTRUCTIONS

```
            MOVFP   Offset, WREG      ; Load WREG with offset to Table
            CALL    Table_LU          ; Call the lookup table
            :
            :

Table_LU    ADDWF   PCL               ; Add Offset to PCL
            RETLW   'A'               ; Returns value in WREG
            RETLW   'B'               ; Returns value in WREG
            :                         ;
```

EXAMPLE 2: PIC18CXXX TABLE LOOK-UP USING THE `RETLW` INSTRUCTIONS

```
            MOVFF   Offset, WREG      ;
            CALL    Table_LU          ; Call the lookup table
            :
            :

Table_LU    BCF     Offset, 7         ; Clear MSb, for rotate to LSb
            RLNCF   Offset, PCL       ; Offset * 2 added to PCL
            RETLW   'A'               ; Returns value in WREG
            RETLW   'B'               ; Returns value in WREG
            :                         ;
```

EXAMPLE 3: USE OF THE '$' PARAMETER

```
          GOTO    $ - 6             ; Replaces  GOTO   $ - 3
          GOTO    $ - 0x2E          ; Replaces  GOTO   $ + 0x17
```

Table Reads and Table Writes

Table Read and Table Write operations have been changed. In the PIC17CXXX architecture, the table pointer register points to the program memory word address. In the PIC18CXXX architecture the table pointer register points to the program memory byte address. This means that the PIC18CXXX is now only operating with 8-bits of data. This allows there to be only one instruction for a Table Read and one instruction for a Table Write. The PIC17CXXX architecture requires two instructions for each, due to operating with 16-bits of data.

Example 4 shows the PIC17CXXX code segment for reading a fixed number of words (WORD_COUNT) into sequential RAM locations using indirect addressing. Example 5 shows the PIC18CXXX code segment for reading a fixed number of bytes (BYTE_COUNT) into sequential RAM locations using indirect addressing.

Migration Impact

The code sections where Table reads and Table writes were implemented would need to be rewritten to address the differences in the implementations.

EXAMPLE 4: PIC17CXXX TABLE READ

```
          MOVLW     WORD_COUNT          ; Load the Word Count value
          MOVWF     CNTR                ;   into CNTR
;
          MOVLW     HIGH(TBL_ADDR)      ; Load the Table Address
          MOVWF     TBLPTRH             ;
          MOVLW     LOW(TBL_ADDR)       ;
          MOVWF     TBLPTRL             ;
          TABLRD    0, 1, DUMMY         ; Dummy read,
                                        ;   Updates TABLATH
                                        ;   Increments TBLPTR
LOOP1     TLRD      1, INDF0            ; Read HI byte in TABLATH
          TABLRD    0, 1, INDF0         ; Read LO byte in TABLATL,
                                        ;   update TABLATH:TABLATL,
                                        ;   and increment TBLPTR
          DECFSZ    CNTR                ; Read Word Count locations
          GOTO      LOOP1               ; Read next word
```

EXAMPLE 5: PIC18CXXX TABLE READ

```
          MOVLW     BYTE_COUNT          ; Load the Byte Count value
          MOVWF     CNTR                ;   into CNTR
;
;;        MOVLW     UPPER(TBL_ADDR)     ; Load the Table Address
;;        MOVWF     TBLPTRU             ;  (on POR TBLPTRU = 0, so
;;                                      ;  loading TBLPTRU is not
;;                                      ;  required for conversions)
          MOVLW     HIGH(TBL_ADDR)      ; Load the Table Address
          MOVWF     TBLPTRH             ;
          MOVLW     LOW(TBL_ADDR)       ;
          MOVWF     TBLPTRL             ;
LOOP1     TBLRD*+                       ; Read value into TABLAT,
                                        ;   Increment TBLPTR
          MOVFF     TABLAT, POSTINC0    ; Copy byte to RAM @ FSR0
                                        ;   Increment FSR0
          DECFSZ    CNTR                ; Read Byte Count locations
          GOTO      LOOP1               ; Read next Byte
```

Interrupts

The interrupt structure of the two families is significantly different. Figure 18 shows a simplified block diagram for the interrupt structures of the two families.

In the PIC17CXXX family, there are four interrupt vector addresses with a priority that is fixed by hardware. In the PIC18CXXX family, there are two interrupt vector addresses. One vector address for High Priority interrupts and one vector address for Low Priority interrupts. The priority of the peripheral interrupt is software programmable.

Table 10 compares the interrupt vector addresses for both the PIC17CXXX and PIC18CXXX families.

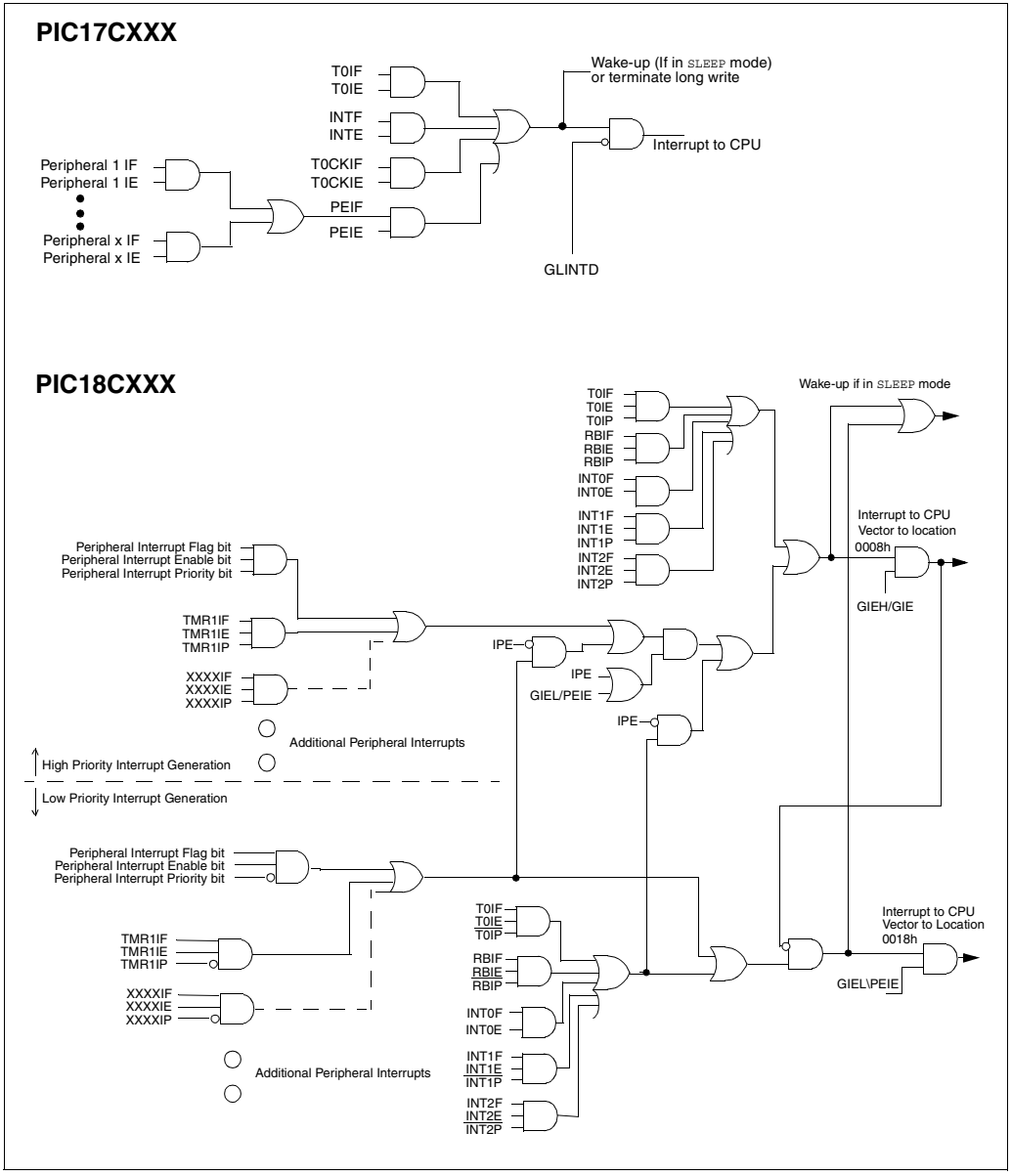
Migration Impact

The code section for interrupt handling would need to be rewritten to address the differences in the implementations. If the PIC17CXXX separate interrupts are desired for a reduction of the interrupt latency, the PIC18CXXX HighPriority/Low Priority vectors may be able to address this.

TABLE 10: INTERRUPT VECTOR ADDRESSES

Location	PIC17CXXX Address		PIC18CXXX Address		Comment
	Word	Byte	Word	Byte	
Reset Vector Address	0000h	N.A.	0000h	0000h	—
INT pin Interrupt Vector Address	0004h	N.A.	—	—	PIC17CXXX must move this code to either the high or low priority interrupt vector address
High Priority Interrupt Vector Address	—	—	0004h	0008h	—
Low Priority Interrupt Vector Address	—	—	000Ch	0018h	—
Timer0 Interrupt Vector Address	0010h	N.A.	—	—	PIC17CXXX must move this code to either the high or low priority interrupt vector address
T0CKI pin Interrupt Vector Address	0018h	N.A.	—	—	PIC17CXXX must move this code to either the high or low priority interrupt vector address
Peripheral Interrupt Vector Address	0020h	N.A.	—	—	For code migration without software enhancements, the code at this address should now be ORG'd to the PIC18CXXX High Priority Interrupt Vector Address (0x018)

FIGURE 18: INTERRUPT STRUCTURE BLOCK DIAGRAMS



Stack

The stack of the PIC17CXXX family is 16 levels deep. When the 17th item is loaded onto the stack, the content of stack level 1 is overwritten (circular buffer). In the PIC18CXXX family, the stack is 31 levels deep. When the 32nd item is loaded onto the stack, the content of stack level 31 is overwritten (stack pointer becomes stuck at 31).

The PIC17CXXX has a stack available bit (STKAV), which indicates if the stack pointer is pointing to the top of stack, or if the stack has rolled over. The PIC18CXXX has 2 bits, which are used to specify if the stack is full (STKFUL) or if underflow (STKUNF) condition has occurred. A configuration bit (STVREN) specifies if these flags generate a device reset.

In the PIC18CXXX, the stack pointer is now memory mapped. This is useful in some applications, such as Real Time Operating Systems (RTOS). Utmost care should be taken if modifying the stack pointer and contents of the stack.

An enhancement of the PIC18CXXX is the implementation of the Fast Register Stack. The Fast Register Stack saves the contents of the WREG, STATUS, and BSR registers. This stack is one level deep for each register. This is useful for saving the status of these registers when you do a subroutine call (if interrupts are disabled), or for interrupts where nesting is not a possibility (do not use with low priority interrupts).

Figure 19 shows the operation of the PIC17CXXX stack, while Figure 20 shows the operation of the PIC18CXXX stack.

Migration Impact

When migrating code from the PIC17CXXX to the PIC18CXXX, one should only need to modify the application software in regards to stack overflows and underflows. If no stack overflow/underflow checking was implemented, then there are no code migration issues due to the hardware stack.

FIGURE 19: PIC17CXXX STACK OPERATION

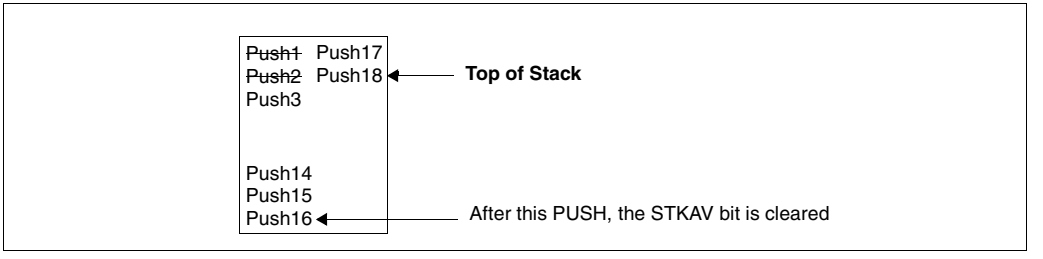
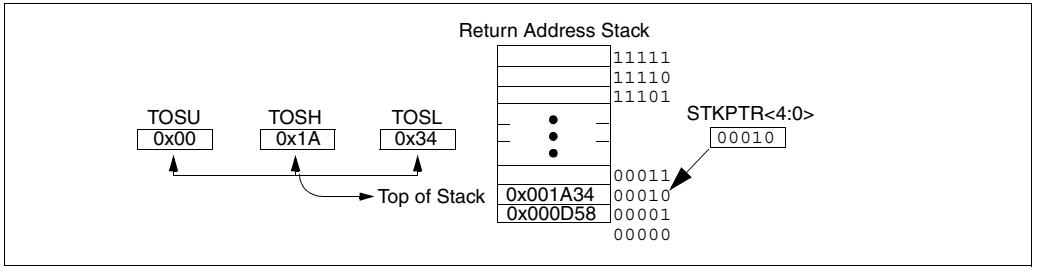


FIGURE 20: PIC18CXXX STACK OPERATION



Indirect Addressing

The PIC17CXXX family has 2 indirect addressing pointers (registers) called FSR0 and FSR1. Each pointer uses an 8-bit register. This allows the indirect addressing to occur anywhere in the selected banks of data memory (SFR bank and GPR bank).

The PIC18CXXX family has 3 indirect addressing pointers (registers) called FSR0, FSR1 and FSR2. Each pointer uses an 12-bit register. This allows the indirect addressing to occur anywhere in the data memory map.

Table 11 shows a comparison of the Indirect Addressing capabilities and operation.

Migration Impact

From the PIC17CXXX code, ensure that the current value of the BSR<3:0> is loaded into the high byte of the FSR register in the PIC18CXXX. This will ensure that the data memory access is in the correct bank.

Also, if any of the FSR automatic increment/decrement features are used (through the manipulation of the PIC17CXXX ALUSTA control bits), the appropriate indirect addressing register needs to be selected in the PIC18CXXX code.

Any indirect accesses to the PIC17CXXX SFRs would require that the PIC18CXXX FSRxH register be loaded with 0x0F. This makes the indirect addresses occur in bank 15.

TABLE 11: INDIRECT ADDRESSING COMPARISON

Feature	PIC17CXXX	PIC18CXXX	Comment
Number of FSR registers	2	3	
FSRx Register Size	8-bits	12-bits	
BSR specifies Bank(s)	Yes	No	PIC17CXXX specifies both SFR and GPR banks
FSR Memory Reach	256 Bytes	4096 Bytes	PIC18CXXX can access entire memory range, PIC17CXXX can access only in selected banks (SFR and GPR banks).
Instruction to load value into FSRx register	No	Yes	`LFSR` instruction is a 2 word 2 cycle instruction
FSR Pre-increment support	No	Yes	PIC18CXXX operation determined by register addressed (register PREINCx)
FSR Post-increment support	Yes	Yes	PIC18CXXX operation determined by register addressed (register POSTINCx). PIC17CXXX operation determined by control bits (FS3:FS2 and FS1:FS0 in register ALUSTA)
FSR Post-decrement support	Yes	Yes	PIC18CXXX operation determined by register addressed (register POSTDECx) PIC17CXXX operation determined by control bits (FS3:FS2 and FS1:FS0 in register ALUSTA)
FSR with Offset support	No	Yes	PIC18CXXX operation determined by register addressed (register PLUSWx)
ALUSTA control bits (FS3:FS2 and FS1:FS0) for Indirect Addressing Operation	Yes	No	PIC18CXXX operation determined by register addressed

LAYOUT

The pinout of the devices will need to be compared on an individual basis. The first PIC18CXXX devices (PIC18CXX2) are design to be footprint/functional compatible with some of the 28- and 40-pin Mid-Range devices. These devices are therefore not footprint compatible with the existing PIC17CXXX devices. This means that a revision of the board layout will be required.

Future PIC18CXXX devices may be footprint compatible, but a pin-by-pin comparison is required to ensure footprint/functional compatibility with the desired PIC17CXXX device. This functional compatibility does not ensure a compatibility with regards to the electrical characteristics of the device (such as I/O pin V_{IL}/V_{IH} characteristics or signal timings).

Migration Impact

A new layout is currently required for all migrations from the PIC17CXXX devices to the PIC18CXXX devices. Future PIC18CXXX devices may be specified that are footprint compatible with PIC17CXXX devices

CODING TECHNIQUES

The conversion process is aided when the initial code was written symbolically. That is, register names, bit names, and address labels are used in the source code as opposed to the hard coded values.

Example 1 shows the technique for using symbols for register and bit definitions, while Example 2 shows labels being used to specify addresses.

EXAMPLE 1: CODE TECHNIQUE #1

```
                BSF     3,2             ; Bad Programmer

                BSF     STATUS, Z       ; Good Programmer
```

EXAMPLE 2: CODE TECHNIQUE #2

```
                GOTO    0x0934          ; Bad Programmer

                GOTO    MY_Routine      ; Good Programmer
                :
                :
MY_Routine      :                       ; This is at address 0x0934
```

EXAMPLE CODE CONVERSION

Appendix A is a code conversion from a code segment found in Application Note AN547, Serial Port Utilities. The code segment was source file SERINT.ASM. The source file includes indications in the comments for each source code line that was changed. This is shown by a comment as follows:

```
;*****.
```

This was done to easily indicate each line that required a change, and specify the change that was implemented to make the source code compatible with the PIC18CXXX assembler.

CONCLUSION

Understanding the issues in a code conversion from one device to another is very important for assuring a smooth conversion process. This document hopefully has given you insight into where to inspect your code during the conversion process.

One of the main architectural goals of the PIC18CXXX family is that of source code compatibility foremost with the PICmicro Mid-Range Architecture and then with the High-End Architecture. The different implementation of peripheral and architectural features are the biggest hurdle. Since some of these peripheral modules and architectural features are implemented differently between the two families, this directly affects the ease of the conversion process. Conversions may require minimal effort in most applications, but there will be features (such as time based functions) that may require a full source code rewrite. Depending on the application of these functions, this rewrite may be relatively minor or fairly involved.

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of source code.

APPENDIX A: EXAMPLE CODE CONVERSION

EXAMPLE 1: CONVERTED SOURCE CODE EXAMPLE

```
;        TITLE   'Serial Interface Routines
;        PROCESSOR     18C452
;
;This is a short program to demonstrate how to transmit and receive
;serial data using the PIC18C452.
;
;A message will be transmitted and routed right back to the processor
;and read. The read information will be saved in an internal buffer.
;
;       Program:          18C_SER.ASM
;       Revision Date:
;                         7-02-99   Conversion to PIC18Cxx2 code.
;                                      Converted from PIC17C42 SERINT.ASM 1-22-97
;                                   as found in AN547 (DS00547C)
;
;
        LIST     P = 18C452

#include    <p18c452.inc>           ;***** Changed the include file
;
;***  These Registers may be remapped to allow the other application software
;***  to take advantage of the access RAM in Bank 0.
;
TX_BUFFER       equ     0x80
RX_BUFFER       equ     0xB0
RXPTR           equ     0x20
TXPTR           equ     0x21
SERFLAG         equ     0x22
RTINUM          equ     0x23
;
;  Status Bits used with user registers
;
TXDONE          equ     0
RXDONE          equ     1
HILOB           equ     2
;
;
```

```
        ORG     0
        goto    start
;
;   Changes ORG directives to point to new High and Low priority interrupts
;   Removed origines to TMRO, T0CKI, and Peripheral interrupts.
;
;        ORG     0x0010          ;vector for rtcc interrupt
                                 ;***** No Longer an separate TMR0 Interrupt Vector Address
;rtcc_int                        ;not used here
;
        ORG     0x0008          ;vector for peripheral interrupt
                                 ;***** Vector Address changed from 0x0020
perf_int
        goto    service_perf    ;service the interrupts
;
        ORG     0x0030
;
;initialize the serial port: baud rate interrupts etc.
init_serial
        clrf    SERFLAG         ;clear all flags
                                 ;***** REMOVED ', F'
;        movlb   0               ;***** REMOVE
        movlw   0x07             ;select 9600 baud
        MOVWF   SPBRG            ;***** Change MOVFP tp MOVWF
        movlw   0x90             ;set up serial pins
        MOVWF   RCSTA            ;***** Change MOVFP tp MOVWF
        clrf    TXSTA            ;setup transmit status
                                 ;***** REMOVED ', F'
;        movlb   1               ;***** REMOVE
        clrf    PIR1             ;clear all interrupts
                                 ;***** REMOVED ', F', Changed PIR -> PIR1
        clrf    PIE1             ;clear all enables
                                 ;***** REMOVED ', F', Changed PIE -> PIE1
        bsf     PIE1,RCIE        ;enable receive interrupt
                                 ;*****    Changed PIE -> PIE1
        movlw   RX_BUFFER        ;set pointer to rx buffer
        MOVWF   RXPTR            ;***** Change MOVPF tp MOVWF
        clrf    INTCON           ;clear all interrupts
                                 ;***** REMOVED ', F', INTSTA -> INTCON
        bsf     INTCON,PEIE      ;enable peripheral ints
                                 ;***** INTSTA -> INTCON
        retfie
;
;start transmission of first two bytes
start_xmit
;        movlb   0               ;***** REMOVE
        bsf     TXSTA,TXEN       ;enable transmit
;        tablrd  1,1,W           ;load latch           ;***** REPLACED
;        tlrd    1,TXREG         ;load high byte       ;***** REPLACED
        TBLRD*+                  ;***** Due to New Implementation of Table Read function
        MOVFF    TABLAT, TXREG   ;***** Due to New Implementation of Table Read function
;        movlb   1               ;***** REMOVE
empty_chk
        btfss   PIR1,TXIF        ;TXBUF empty?
                                 ;*****    Changed PIR -> PIR1
        goto    empty_chk        ;no then keep checking
;        movlb   0               ;***** REMOVE
;        tablrd  0,1,TXREG        ;load lo byte         ;***** REPLACED
        TBLRD*+                  ;***** Due to New Implementation of Table Read function
        MOVFF   TABLAT, TXREG    ;***** Due to New Implementation of Table Read function
;        movlb   1               ;***** REMOVE
        bsf     PIE1,TXIE        ;enable transmit interrupts
                                 ;*****    Changed PIE -> PIE1
        bsf     SERFLAG,HILOB    ;set up next for high byte
        return
;
```

```
;
        PAGE
;
service_perf
;check for transmit or receive interrupts only
        btfsc   PIR1,RCIF       ;RX buffer full?
                                ;*****    Changed PIR -> PIR1
        goto    service_recv    ;yes then service
        btfss   PIR1,TXIF       ;TX buffer empty?
                                ;*****    Changed PIR -> PIR1
        goto    exit_perf       ;no, ignore other int.
service_xmt
        btfsc   SERFLAG,TXDONE  ;all done?
        goto    exit_perf       ;yes then quit
        btfsc   SERFLAG,HILOB   ;if clr, do low byte
        goto    rd_hi           ;else read high byte
;        tablrd  0,1,W           ;read lo                       ;***** REPLACE
        TBLRD*+                 ;***** Due to New Implementation of Table Read function
        goto    sx_cont         ;continue
rd_hi
;        tlrd    1,W             ;read high byte                ;***** REPLACE
        TBLRD*+                 ;***** Due to New Implementation of Table Read function
sx_cont
        btg     SERFLAG,HILOB   ;toggle flag
;        movlb   0               ;***** REMOVE
;        MOVPF   TXREG           ;***** REMOVE, TXREG loaded by next instruction
        MOVFF   TABLAT, TXREG   ;***** Due to New Implementation of Table Read function
        tstfsz  W               ;last byte?
        goto    exit_perf       ;no then cont
end_xmt                         ;else end transmit
;        movlb   1               ;***** REMOVE
        bcf     PIE1,TXIE       ;disable tx interrupt
                                ;******     Changed PIE -> PIE1
        bsf     SERFLAG,TXDONE  ;set done flag
exit_perf
;        bcf     INTSTA,PEIF     ;***** REMOVE,   clear peripheral int
                                ;*****    This instruction was never needed
        retfie
;
service_recv
        btfsc   SERFLAG,RXDONE  ;RX complete?
        goto    exit_perf       ;exit int
        MOVFF   RXPTR, FSR0L    ;***** Change MOVFP to MOVFF and FSR0 to FSR0L
;        movlb   0               ;***** REMOVE
        MOVFF   RCREG,INDF0     ;***** Change MOVFP to MOVFF
        clrf    WREG            ;clr W
                                ;***** REMOVED ', F'
        cpfsgt  INDF0           ;value = 0?
        goto    end_recv        ;yes then end
        incf    FSR0L, F        ;inc pointer
                                ;***** REMOVED ', F', Changed FSR0 to FSR0L and specified destination
        MOVFF   FSR0L, RXPTR    ;***** Change MOVFP to MOVFF and FSR0 to FSR0L
        goto    exit_perf       ;return from int
end_recv
        bsf     SERFLAG,RXDONE  ;set flag
        clrf    INTCON          ;clear all int
                                ;***** REMOVED ', F', INTSTA -> INTCON
;        movlb   1               ;***** REMOVE
        bcf     PIE1,RCIE       ;disable rx interrupts
                                ;*****     Changed PIE -> PIE1
        goto    exit_perf       ;return
        PAGE
```

```
;
start
        clrf    FSR1L           ;assign FSR1 as S.P.
                                ;***** REMOVED ', F' and Changed FSR0 to FSR0L
        decf    FSR1L, F        ;       /
                                ;***** REMOVED ', F', Changed FSR0 to FSR0L and
                                ;*****      specified destination
        movlw   0x20            ;clear ram space
        MOVWF   FSR0L           ;***** Change MOVFP to MOVWF and FSR0 to FSR0L
start1
        clrf    INDF0              ;clear ram
                                   ;***** REMOVED ', F'
        incfsz  FSR0L, F           ;inc and skip if done
                                   ;***** REMOVED ', F', Changed FSR0 to FSR0L and
                                   ;*****      specified destination
        goto    start1
        call    init_serial        ;initialize serial port
        movlw   LOW   MESSAGE      ;load table pointer
        MOVWF   TBLPTRL            ;***** Change MOVPF tp MOVWF
        movlw   HIGH  MESSAGE      ;     /
        MOVWF   TBLPTRH            ;***** Change MOVPF tp MOVWF
        CLRF    TBLPTRU            ;***** ADDED this instruction due to larger memory
                                   ;*****     space of PIC18Cxxx Architecture
        call    start_xmit         ;start transmission
chk_end
        btfss   SERFLAG,RXDONE     ;receive all?
        goto    chk_end            ;no then keep checking
;
loop    goto    loop               ;spin wheel
;
        ORG     0x100
MESSAGE
        DATA    "The code is: Tea for the Tillerman"
        DATA    0
;
;
        END
```

NOTES:

How to Implement ICSP™ Using PIC16CXXX OTP MCUs

Author: *Rodger Richey*
Microchip Technology Inc.

INTRODUCTION

In-Circuit Serial Programming (ICSP™) is a great way to reduce your inventory overhead and time-to-market for your product. By assembling your product with a blank Microchip microcontroller (MCU), you can stock one design. When an order has been placed, these units can be programmed with the latest revision of firmware, tested, and shipped in a very short time. This method also reduces scrapped inventory due to old firmware revisions. This type of manufacturing system can also facilitate quick turnarounds on custom orders for your product.

Most people would think to use ICSP with PICmicro™ OTP MCUs only on an assembly line where the device is programmed once. However, there is a method by which an OTP device can be programmed several times depending on the size of the firmware. This method, explained later, provides a way to field upgrade your firmware in a way similar to EEPROM- or Flash-based devices.

HOW DOES ICSP WORK?

Now that ICSP appeals to you, what steps do you take to implement it in your application? There are three main components of an ICSP system: Application Circuit, Programmer and Programming Environment.

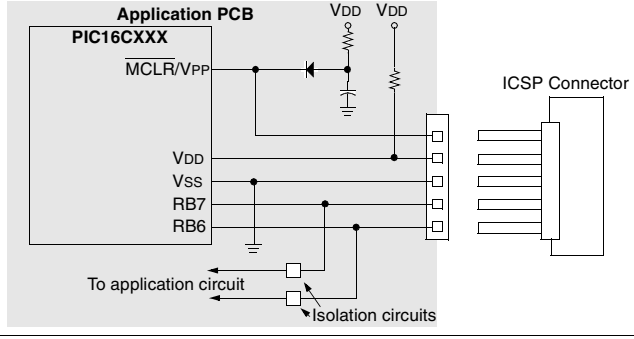
Application Circuit

The application circuit must be designed to allow all the programming signals to be directly connected to the PICmicro. Figure 1 shows a typical circuit that is a starting point for when designing with ICSP. The application must compensate for the following issues:

1. Isolation of the $\overline{MCLR}$/VPP pin from the rest of the circuit.
2. Isolation of pins RB6 and RB7 from the rest of the circuit.
3. Capacitance on each of the VDD, $\overline{MCLR}$/VPP, RB6, and RB7 pins.
4. Minimum and maximum operating voltage for VDD.
5. PICmicro Oscillator.
6. Interface to the programmer.

The $\overline{MCLR}$/VPP pin is normally connected to an RC circuit. The pull-up resistor is tied to VDD and a capacitor is tied to ground. This circuit can affect the operation of ICSP depending on the size of the capacitor. It is, therefore, recommended that the circuit in Figure 1 be used when an RC is connected to $\overline{MCLR}$/VPP. The diode should be a Schottky-type device. Another issue with $\overline{MCLR}$/VPP is that when the PICmicro device is programmed, this pin is driven to approximately 13V and also to ground. Therefore, the application circuit must be isolated from this voltage provided by the programmer.

FIGURE 1: TYPICAL APPLICATION CIRCUIT



Pins RB6 and RB7 are used by the PICmicro for serial programming. RB6 is the clock line and RB7 is the data line. RB6 is driven by the programmer. RB7 is a bi-directional pin that is driven by the programmer when programming, and driven by the PICmicro when verifying. These pins must be isolated from the rest of the application circuit so as not to affect the signals during programming. You must take into consideration the output impedance of the programmer when isolating RB6 and RB7 from the rest of the circuit. This isolation circuit must account for RB6 being an input on the PICmicro, and for RB7 being bi-directional (can be driven by both the PICmicro and the programmer). For instance, PRO MATE® II has an output impedance of 1k¾. If the design permits, these pins should not be used by the application. This is not the case with most applications so it is recommended that the designer evaluate whether these signals need to be buffered. As a designer, you must consider what type of circuitry is connected to RB6 and RB7 and then make a decision on how to isolate these pins. Figure 1 does not show any circuitry to isolate RB6 and RB7 on the application circuit because this is very application dependent.

The total capacitance on the programming pins affects the rise rates of these signals as they are driven out of the programmer. Typical circuits use several hundred microfarads of capacitance on VDD which helps to dampen noise and ripple. However, this capacitance requires a fairly strong driver in the programmer to meet the rise rate timings for VDD. Most programmers are designed to simply program the PICmicro itself and don't have strong enough drivers to power the application circuit. One solution is to use a driver board between the programmer and the application circuit. The driver board requires a separate power supply that is capable of driving the VPP and VDD pins with the correct rise rates and should also provide enough current to power the application circuit. RB6 and RB7 are not buffered on this schematic but may require buffering depending upon the application. A sample driver board schematic is shown in Appendix A.

Note:	The driver board design MUST be tested in the user's application to determine the effects of the application circuit on the programming signals timing. Changes may be required if the application places a significant load on VDD, VPP, RB6 OR RB7.

The Microchip programming specification states that the device should be programmed at 5V. Special considerations must be made if your application circuit operates at 3V only. These considerations may include totally isolating the PICmicro during programming. The other issue is that the device must be verified at the minimum and maximum voltages at which the application circuit will be operating. For instance, a battery operated system may operate from three 1.5V cells giving an operating voltage range of 2.7V to 4.5V. The programmer must program the device at 5V and must verify the program memory contents at both 2.7V and 4.5V to ensure that proper programming margins have been achieved. This ensures the PICmicro option over the voltage range of the system.

This final issue deals with the oscillator circuit on the application board. The voltage on MCLR/VPP must rise to the specified program mode entry voltage before the device executes any code. The crystal modes available on the PICmicro are not affected by this issue because the Oscillator Start-up Timer waits for 1024 oscillations before any code is executed. However, RC oscillators do not require any startup time and, therefore, the Oscillator Startup Timer is not used. The programmer must drive MCLR/VPP to the program mode entry voltage before the RC oscillator toggles four times. If the RC oscillator toggles four or more times, the program counter will be incremented to some value X. Now when the device enters programming mode, the program counter will not be zero and the programmer will start programming your code at an offset of X. There are several alternatives that can compensate for a slow rise rate on MCLR/VPP. The first method would be to not populate the R, program the device, and then insert the R. The other method would be to have the programming interface drive the OSC1 pin of the PICmicro to ground while programming. This will prevent any oscillations from occurring during programming.

Now all that is left is how to connect the application circuit to the programmer. This depends a lot on the programming environment and will be discussed in that section.

Programmer

The second consideration is the programmer. PIC16CXXX MCUs only use serial programming and therefore all programmers supporting these devices will support ICSP. One issue with the programmer is the drive capability. As discussed before, it must be able to provide the specified rise rates on the ICSP signals and also provide enough current to power the application circuit. Appendix A shows an example driver board. This driver schematic does not show any buffer circuitry for RB6 and RB7. It is recommended that an evaluation be performed to determine if buffering is required. Another issue with the programmer is what VDD levels are used to verify the memory contents of the PICmicro. For instance, the PRO MATE II verifies program memory at the minimum and maximum VDD levels for the specified device and is therefore considered a production quality programmer. On the other hand, the PICSTART® Plus only verifies at 5V and is for prototyping use only. The Microchip programming specifications state that the program memory contents should be verified at both the minimum and maximum VDD levels that the application circuit will be operating. This implies that the application circuit must be able to handle the varying VDD voltages.

There are also several third party programmers that are available. You should select a programmer based on the features it has and how it fits into your programming environment. The *Microchip Development Systems Ordering Guide* (DS30177) provides detailed information on all our development tools. The *Microchip Third Party Guide* (DS00104) provides information on all of our third party tool developers. Please consult these two references when selecting a programmer. Many options exist including serial or parallel PC host connection, stand-alone operation, and single or gang programmers. Some of the third party developers include Advanced Transdata Corporation, BP Microsystems, Data I/O, Emulation Technology and Logical Devices.

Programming Environment

The programming environment will affect the type of programmer used, the programmer cable length, and the application circuit interface. Some programmers are well suited for a manual assembly line while others are desirable for an automated assembly line. You may want to choose a gang programmer to program multiple systems at a time.

The physical distance between the programmer and the application circuit affects the load capacitance on each of the programming signals. This will directly affect the drive strength needed to provide the correct signal rise rates and current. This programming cable must also be as short as possible and properly terminated and shielded, or the programming signals may be corrupted by ringing or noise.

Finally, the application circuit interface to the programmer depends on the size constraints of the application circuit itself and the assembly line. A simple header can be used to interface the application circuit to the programmer. This might be more desirable for a manual assembly line where a technician plugs the programmer cable into the board. A different method is the use of spring loaded test pins (commonly referred to as pogo pins). The application circuit has pads on the board for each of the programming signals. Then there is a fixture that has pogo pins in the same configuration as the pads on the board. The application circuit or fixture is moved into position such that the pogo pins come into contact with the board. This method might be more suitable for an automated assembly line.

After taking into consideration the issues with the application circuit, the programmer, and the programming environment, anyone can build a high quality, reliable manufacturing line based on ICSP.

Other Benefits

ICSP provides other benefits, such as calibration and serialization. If program memory permits, it would be cheaper and more reliable to store calibration constants in program memory instead of using an external serial EEPROM. For example, your system has a thermistor which can vary from one system to another. Storing some calibration information in a table format allows the micrcontroller to compensate in software for external component tolerances. System cost can be reduced without affecting the required performance of the system by using software calibration techniques. But how does this relate to ICSP? The PICmicro has already been programmed with firmware that performs a calibration cycle. The calibration data is transferred to a calibration fixture. When all calibration data has been transferred, the fixture places the PICmicro in programming mode and programs the PICmicro with the calibration data. Application note *AN656, In-Circuit Serial Programming of Calibration Parameters Using a PICmicro Microcontroller*, shows exactly how to implement this type of calibration data programming.

The other benefit of ICSP is serialization. Each individual system can be programmed with a unique or random serial number. One such application of a unique serial number would be for security systems. A typical system might use DIP switches to set the serial number. Instead, this number can be burned into program memory, thus reducing the overall system cost and lowering the risk of tampering.

Field Programming of PICmicro OTP MCUs

An OTP device is not normally capable of being reprogrammed, but the PICmicro architecture gives you this flexibility provided the size of your firmware is at least half that of the desired device and the device is not code protected. If your target device does not have enough program memory, Microchip provides a wide spectrum of devices from 0.5K to 8K program memory with the same set of peripheral features that will help meet the criteria.

The PIC16CXXX microcontrollers have two vectors, reset and interrupt, at locations 0x0000 and 0x0004. When the PICmicro encounters a reset or interrupt condition, the code located at one of these two locations in program memory is executed. The first listing of Example 1-1 shows the code that is first programmed into the PICmicro. The second listing of Example 1-1 shows the code that is programmed into the PICmicro for the second time.

EXAMPLE 1-1: PROGRAMMING CYCLE LISTING FILES

```
       First Program Cycle                    Second Program Cycle
_____________________________________________________________________________________________
Prog    Opcode   Assembly                  |ProgOpcodeAssembly
Mem              Instruction               |Mem      Instruction
---------------------------------------------------------------------------------------------
0000    2808     goto  Main    ;Main loop  |         0000     0000           nop
0001    3FFF     <blank>       ;at 0x0008  |  0001   2860     goto  Main     ;Main now
0002    3FFF     <blank>                   |  0002   3FFF     <blank>        ;at 0x0060
0003    3FFF     <blank>                   |  0003   3FFF     <blank>
0004    2848     goto  ISR     ;ISR at     |  0004   0000     nop
0005    3FFF     <blank>       ;0x0048     |  0005   28A8     goto  ISR      ;ISR now at
0006    3FFF     <blank>                   |  0006   3FFF     <blank>        ;0x00A8
0007    3FFF     <blank>                   |  0007   3FFF     <blank>
0008    1683     bsf STATUS,RP0            |          0008    1683           bsf STATUS,RP0
0009    3007     movlw 0x07                |  0009   3007     movlw 0x07
000A    009F     movwf ADCON1              |  000A   009F     movwf ADCON1
 .                                         |   .
 .                                         |   .
 .                                         |   .
0048    1C0C     btfss PIR1,RBIF           |  0048   1C0C     btfss PIR1,RBIF
0049    284E     goto  EndISR              |  0049   284E     goto  EndISR
004A    1806     btfsc PORTB,0             |  004A   1806     btfsc PORTB,0
 .                                         |   .
 .                                         |   .
 .                                         |   .
0060    3FFF     <blank>                   |  0060   1683     bsf   STATUS,RP0
0061    3FFF     <blank>                   |  0061   3005     movlw 0x05
0062    3FFF     <blank>                   |  0062   009F     movwf ADCON1
 .                                         |   .
 .                                         |   .
 .                                         |   .
00A8    3FFF     <blank>                   |  00A8   1C0C     btfss PIR1,RBIF
00A9    3FFF     <blank>                   |  00A9   28AE     goto  EndISR
00AA    3FFF     <blank>                   |  00AA   1806     btfsc PORTB,0
 .                                         |   .
 .                                         |   .
 .                                         |   .
---------------------------------------------------------------------------------------------
```

The example shows that to program the PICmicro a second time the memory location 0x0000, originally `goto Main` (0x2808), is reprogrammed to all 0's which happens to be a `nop` instruction. This location cannot be reprogrammed to the new opcode (0x2860) because the bits that are 0's cannot be reprogrammed to 1's, only bits that are 1's can be reprogrammed to 0's. The next memory location 0x0001 was originally blank (all 1's) and now becomes a `goto Main` (0x2860). When a reset condition occurs, the PICmicro executes the instruction at location 0x0000 which is the `nop`, a completely benign instruction, and then executes the `goto Main` to start the execution of code. The example also shows that all program memory locations after 0x005A are blank in the original program so that the second time the PICmicro is programmed, the revised code can be programmed at these locations. The same descriptions can be given for the interrupt vector at location 0x0004.

This method changes slightly for PICmicros with >2K words of program memory. Each of the `goto Main` and `goto ISR` instructions are replaced by the following code segments due to paging on devices with >2K words of program memory.

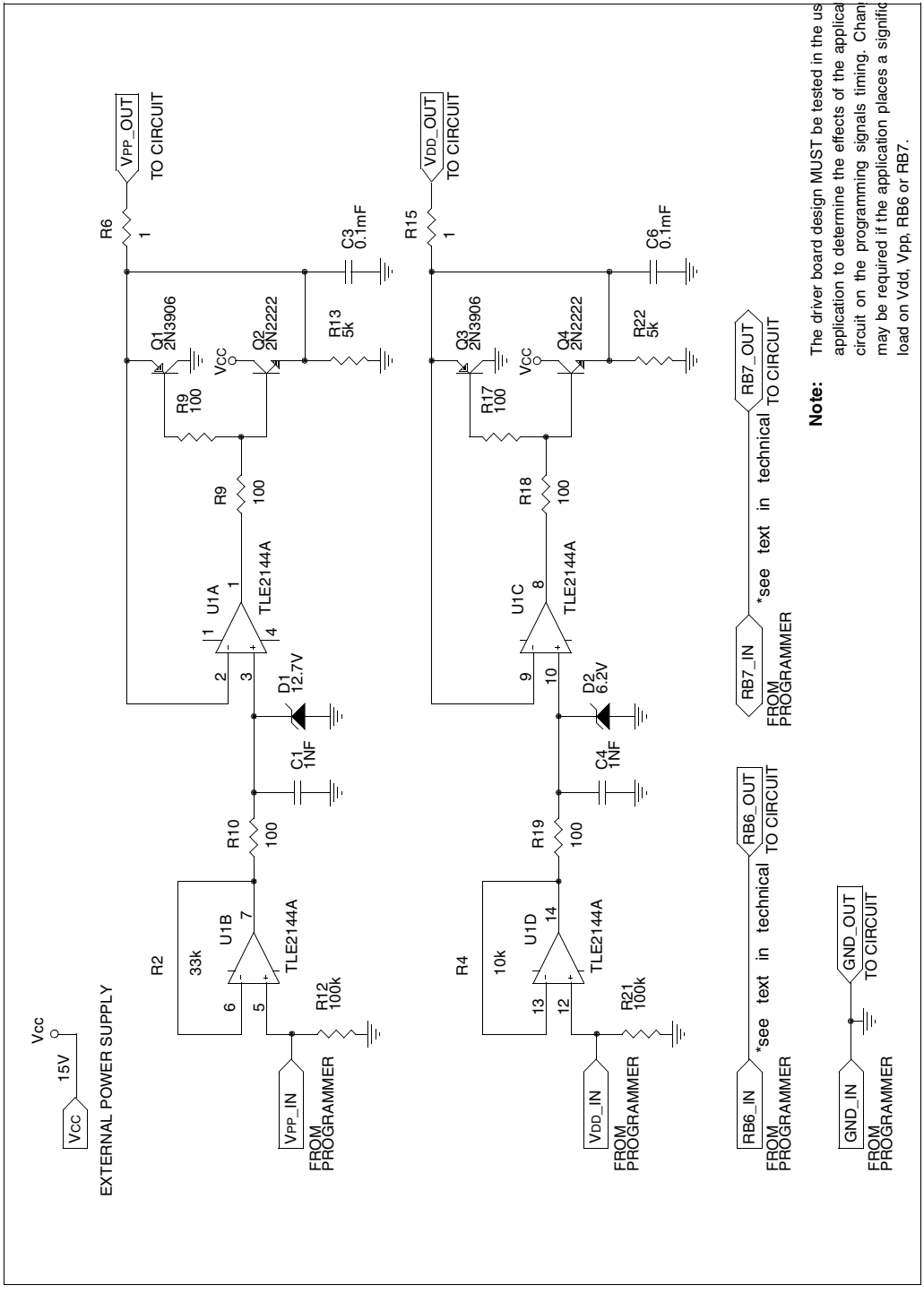
```
movlw <page>movlw <page>
movwf PCLATHmovwf PCLATH
goto  Main goto  ISR
```

Now your one time programmable PICmicro is exhibiting more EEPROM- or Flash-like qualities.

CONCLUSION

Microchip Technology Inc. is committed to supporting your ICSP needs by providing you with our many years of experience and expertise in developing ICSP solutions. Anyone can create a reliable ICSP programming station by coupling our background with some forethought to the circuit design and programmer selection issues previously mentioned. Your local Microchip representative is available to answer any questions you have about the requirements for ICSP.

APPENDIX A: SAMPLE DRIVER BOARD SCHEMATIC



How to Implement ICSP™ Using PIC17CXXX OTP MCUs

Author: *Stan D'Souza*
Microchip Technology Inc.

INTRODUCTION

PIC17CXXX microcontroller (MCU) devices can be serially programmed using an RS-232 or equivalent serial interface. As shown in Figure 1, using just three pins, the PIC17CXXX can be connected to an external interface and programmed. In-Circuit Serial Programming (ICSP™) allows for a greater flexibility in an application as well as a faster time to market for the user's product.

This technical brief will demonstrate the practical aspects associated with ICSP using the PIC17CXXX. It will also demonstrate some key capabilities of OTP devices when used in conjunction with ICSP.

Implementation

The PIC17CXXX devices have special instructions, which enables the user to program and read the PIC17CXXX's program memory. The instructions are `TABLWT` and `TLWT` which implement the program memory write operation and `TABLRD` and `TLRD` which perform the program memory read operation. For more details, please check the *In-Circuit Serial Programming for PIC17CXXX OTP Microcontrollers Specification* (DS30273), PIC17C4X data sheet (DS30412) and PIC17C75X data sheet (DS30264).

When doing ICSP, the PIC17CXXX runs a boot code, which configures the USART port and receives data serially through the RX line. This data is then programmed at the address specified in the serial data string. A high voltage (about 13V) is required for the EPROM cell to get programmed, and this is usually supplied by the programming header as shown in Figure 1 and Figure 2. The PIC17CXXX's boot code enables and disables the high voltage line using a dedicated I/O line.

FIGURE 1: PIC17CXXX IN-CIRCUIT SERIAL PROGRAMMING USING TABLE WRITE INSTRUCTIONS

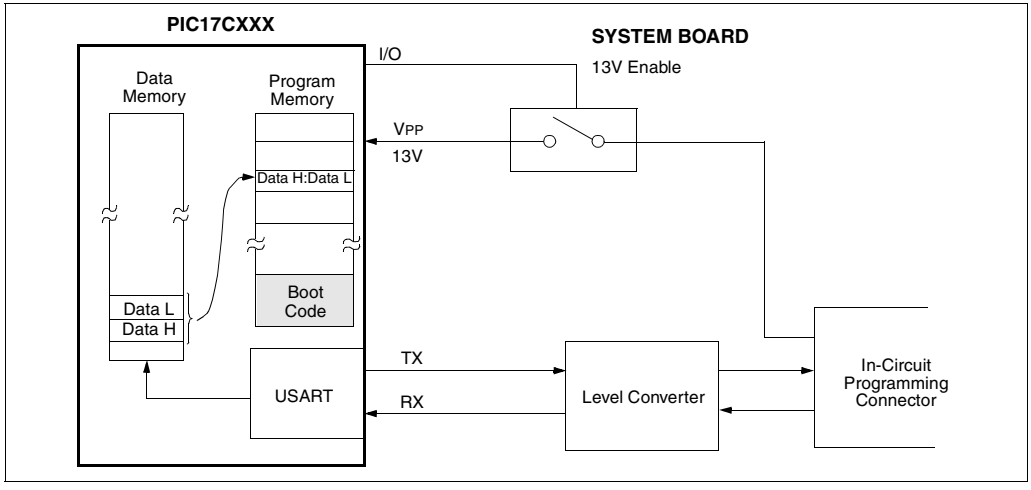
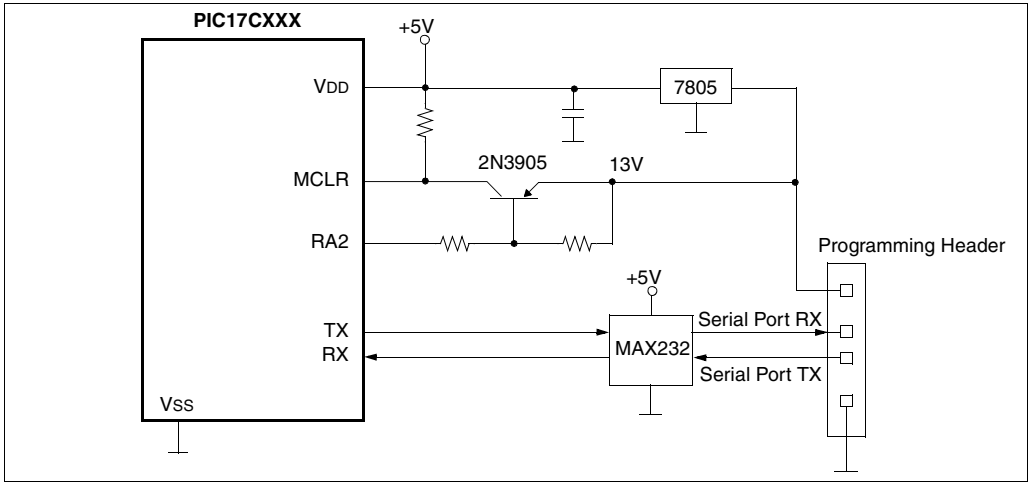


FIGURE 2: PIC17CXXX IN-CIRCUIT SERIAL PROGRAMMING SCHEMATIC



ICSP Boot Code

The boot code is normally programmed, into the PIC17CXXX device using a PRO MATE® or PICSTART® Plus or any third party programmer. As depicted in the flowchart in Figure 4, on power-up, or a reset, the program execution always vectors to the boot code. The boot code is normally located at the bottom of the program memory space e.g. 0x700 for a PIC17C42A (Figure 3).

Several methods could be used to reset the PIC17CXXX when the ICSP header is connected to the system board. The simplest method, as shown in Figure 2, is to derive the system 5V, from the 13V supplied by the ICSP header. It is quite common in manufacturing lines, to have system boards programmed with only the boot code ready and available for testing, calibration or final programming. The ICSP header would thus supply the 13V to the system and this 13V would then be stepped down to supply the 5V required to power the system. Please note that the 13V supply should have enough drive capability to supply power to the system as well as maintain the programming voltage of 13V.

The first action of the boot code (as shown in flowchart Figure 4) is to configure the USART to a known baud rate and transmit a request sequence to the ICSP host system. The host immediately responds with an acknowledgment of this request. The boot code then gets ready to receive ICSP data. The host starts sending the data and address byte sequences to the PIC17CXXX. On receiving the address and data information, the 16-bit address is loaded into the TBLPTR registers and the 16-bit data is loaded into the TABLAT registers. The RA2 pin is driven low to enable 13V at $\overline{\text{MCLR}}$. The PIC17CXXX device then executes a table write instruction. This instruction in turn causes a long write operation, which disables further code execution. Code execution is resumed when an internal interrupt occurs. This delay ensures that the programming pulse width of 1 ms (max.) is met. Once a location is written, RA2 is driven high to disable further writes and a verify operation is done using the Table read instruction. If the result is good, an acknowledge is sent to the host. This process is repeated till all desired locations are programmed.

In normal operation, when the ICSP header is not connected, the boot code would still execute and the PIC17CXXX would send out a request to the host. However it would not get a response from the host, so it would abort the boot code and start normal code execution.

FIGURE 3: BOOT CODE EXAMPLE FOR PIC17C42A

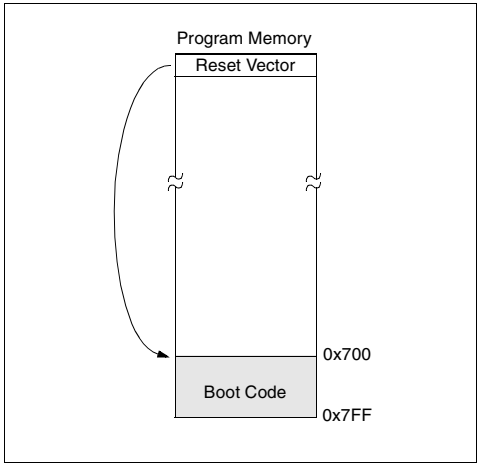
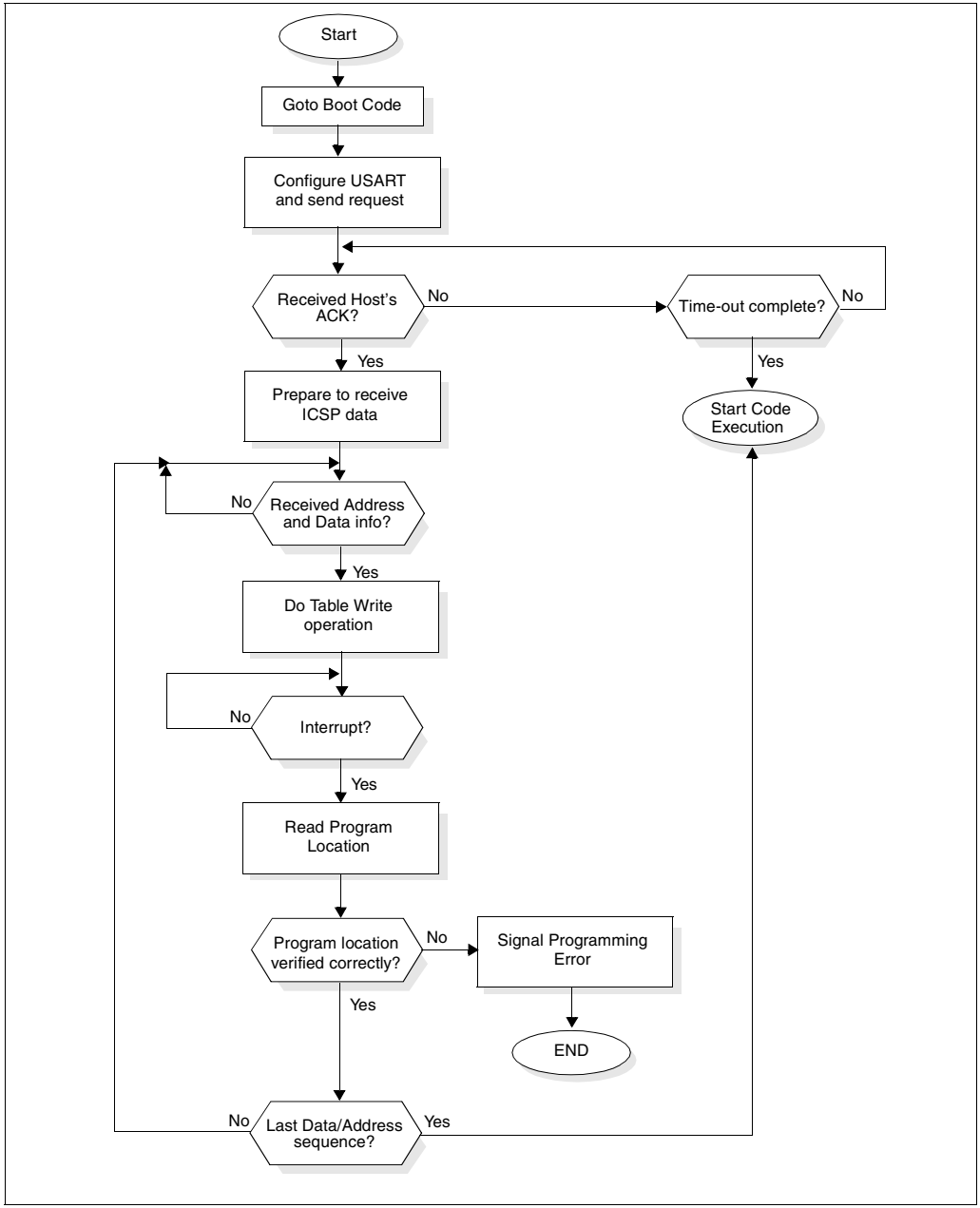


FIGURE 4: FLOWCHART FOR ICSP BOOT CODE



USING THE ICSP FEATURE ON PIC17CXXX OTP DEVICES

The ICSP feature is a very powerful tool when used in conjunction with OTP devices.

Saving Calibration Information Using ICSP

One key use of ICSP is to store calibration constants or parameters in program memory. It is quite common to interface a PIC17CXXX device to a sensor. Accurate, pre-calibrated sensors can be used, but they are more expensive and have long lead times. Un-calibrated sensors on the other hand are inexpensive and readily available. The only caveat is that these sensors have to be calibrated in the application. Once the calibration constants have been determined, they would be unique to a given system, so they have to be saved in program memory. These calibration parameters/constants can then be retrieved later during program execution and used to improve the accuracy of low cost un-calibrated sensors. ICSP thus offers a cost reduction path for the end user in the application.

Saving Field Calibration Information Using ICSP

Sensors typically tend to drift and lose calibration over time and usage. One expensive solution would be to replace the sensor with a new one. A more cost effective solution however, is to re-calibrated the system and save the new calibration parameter/constants into the PIC17CXXX devices using ICSP. The user program however has to take into account certain issues:

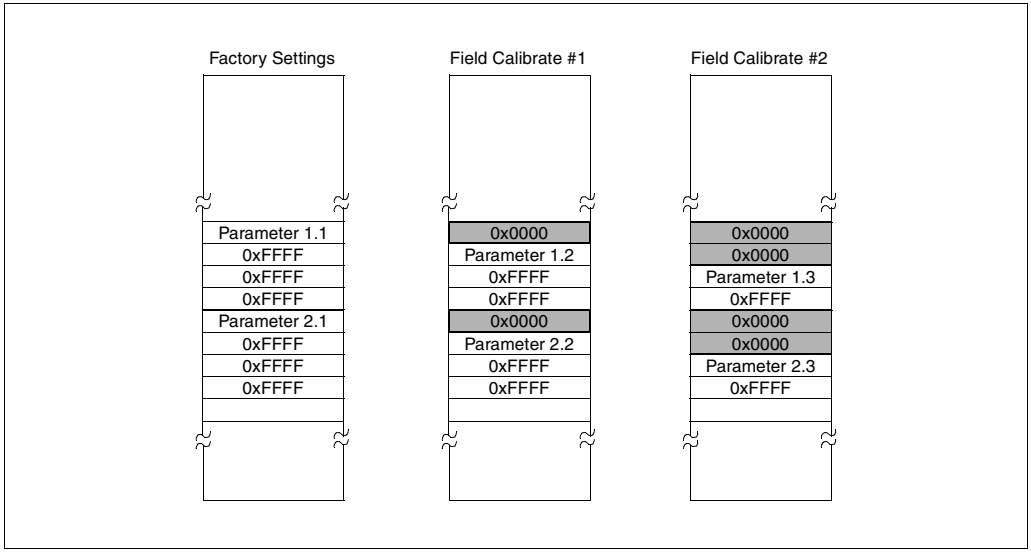
1. Un-programmed or blank locations have to be reserved at each calibration constant location in order to save new calibration parameters/constants.
2. The old calibration parameters/constants are all programmed to 0, so the user program will have to be "intelligent" and differentiate between blank (0xFFFF), zero (0x0000), and programmed locations.

Figure 5 shows how this can be achieved.

Programming Unique Serial Numbers Using ICSP

There are applications where each system needs to have a unique and sometimes random serial number. Example: security devices. One common solution is to have a set of DIP switches which are then set to a unique value during final test. A more cost effective solution however would be to program unique serial numbers into the device using ICSP. The user application can thus eliminate the need for DIP switches and subsequently reduce the cost of the system.

FIGURE 5: FIELD CALIBRATION USING ICSP



Code Updates in the Field Using ICSP

With fast time to market it is not uncommon to see application programs which need to be updated or corrected for either enhancements or minor errors/bugs. If ROM parts were used, updates would be impossible and the product would either become outdated or recalled from the field. A more cost effective solution is to use OTP devices with ICSP and program them in the field with the new updates. Figure 6 shows an example where the user has allowed for one field update to his program.

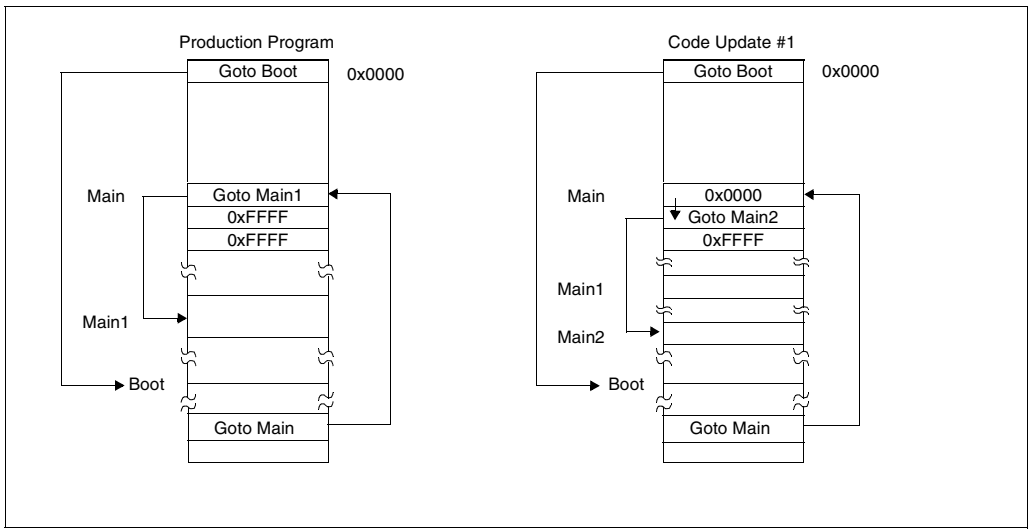
Here are some of the issues which need to be addressed:

1. The user has to reserve sufficient blank memory to fit his updated code.
2. At least one blank location needs to be saved at the reset vector as well as for all the interrupts.
3. Program all the old "goto" locations (located at the reset vector and the interrupts vectors) to 0 so that these instructions execute as NOPs.
4. Program new "goto" locations (at the reset vector and the interrupt vectors) just below the old "goto" locations.
5. Finally, program the new updated code in the blank memory space.

CONCLUSION

ICSP is a very powerful feature available on the PIC17CXXX devices. It offers tremendous design flexibility to the end user in terms of saving calibration constants and updating code in final production as well as in the field, thus helping the user design a low-cost and fast time-to-market product.

FIGURE 6: CODE UPDATES USING ICSP



NOTES:

How to Implement ICSP™ Using PIC16F8X FLASH MCUs

Author: *Rodger Richey*
Microchip Technology Inc.

INTRODUCTION

In-Circuit Serial Programming (ICSP™) with PICmicro® FLASH microcontrollers (MCU) is not only a great way to reduce your inventory overhead and time-to-market for your product, but also to easily provide field upgrades of firmware. By assembling your product with a Microchip FLASH-based MCU, you can stock the shelf with one system. When an order has been placed, these units can be programmed with the latest revision of firmware, tested, and shipped in a very short time. This type of manufacturing system can also facilitate quick turnarounds on custom orders for your product. You don't have to worry about scrapped inventory because of the FLASH-based program memory. This gives you the advantage of upgrading the firmware at any time to fix those "features" that pop up from time to time.

HOW DOES ICSP WORK?

Now that ICSP appeals to you, what steps do you take to implement it in your application? There are three main components of an ICSP system.

These are the: Application Circuit, Programmer and Programming Environment.

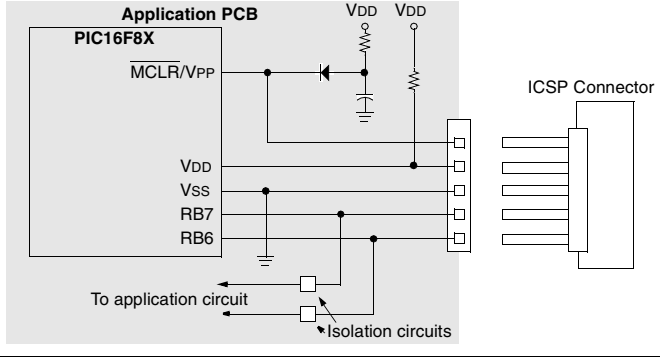
Application Circuit

The application circuit must be designed to allow all the programming signals to be directly connected to the PICmicro. Figure 1 shows a typical circuit that is a starting point for when designing with ICSP. The application must compensate for the following issues:

1. Isolation of the $\overline{MCLR}$/VPP pin from the rest of the circuit.
2. Isolation of pins RB6 and RB7 from the rest of the circuit.
3. Capacitance on each of the VDD, $\overline{MCLR}$/VPP, RB6, and RB7 pins.
4. Minimum and maximum operating voltage for VDD.
5. PICmicro Oscillator.
6. Interface to the programmer.

The $\overline{MCLR}$/VPP pin is normally connected to an RC circuit. The pull-up resistor is tied to VDD and a capacitor is tied to ground. This circuit can affect the operation of ICSP depending on the size of the capacitor. It is, therefore, recommended that the circuit in Figure 1 be used when an RC is connected to $\overline{MCLR}$/VPP. The diode should be a Schottky-type device. Another issue with $\overline{MCLR}$/VPP is that when the PICmicro device is programmed, this pin is driven to approximately 13V and also to ground. Therefore, the application circuit must be isolated from this voltage provided by the programmer.

FIGURE 1: TYPICAL APPLICATION CIRCUIT



PRO MATE, PICSTART and PICmicro are registered trademarks and ICSP is a trademark of Microchip Technology Inc.

Pins RB6 and RB7 are used by the PICmicro for serial programming. RB6 is the clock line and RB7 is the data line. RB6 is driven by the programmer. RB7 is a bi-directional pin that is driven by the programmer when programming, and driven by the PICmicro when verifying. These pins must be isolated from the rest of the application circuit so as not to affect the signals during programming. You must take into consideration the output impedance of the programmer when isolating RB6 and RB7 from the rest of the circuit. This isolation circuit must account for RB6 being an input on the PICmicro and for RB7 being bi-directional (can be driven by both the PICmicro and the programmer). For instance, PRO MATE® II has an output impedance of 1k¾. If the design permits, these pins should not be used by the application. This is not the case with most applications so it is recommended that the designer evaluate whether these signals need to be buffered. As a designer, you must consider what type of circuitry is connected to RB6 and RB7 and then make a decision on how to isolate these pins. Figure 1 does not show any circuitry to isolate RB6 and RB7 on the application circuit because this is very application dependent.

The total capacitance on the programming pins affects the rise rates of these signals as they are driven out of the programmer. Typical circuits use several hundred microfarads of capacitance on VDD which helps to dampen noise and ripple. However, this capacitance requires a fairly strong driver in the programmer to meet the rise rate timings for VDD. Most programmers are designed to simply program the PICmicro itself and don't have strong enough drivers to power the application circuit. One solution is to use a driver board between the programmer and the application circuit. The driver board requires a separate power supply that is capable of driving the VPP and VDD pins with the correct rise rates and should also provide enough current to power the application circuit. RB6 and RB7 are not buffered on this schematic but may require buffering depending upon the application. A sample driver board schematic is shown in Appendix A.

Note:	The driver board design MUST be tested in the user's application to determine the effects of the application circuit on the programming signals timing. Changes may be required if the application places a significant load on Vdd, Vpp, RB6 or RB7.

The Microchip programming specification states that the device should be programmed at 5V. Special considerations must be made if your application circuit operates at 3V only. These considerations may include totally isolating the PICmicro during programming. The other issue is that the device must be verified at the minimum and maximum voltages at which the application circuit will be operating. For instance, a battery operated system may operate from three 1.5V cells giving an operating voltage range of 2.7V to 4.5V. The programmer must program the device at 5V and must verify the program memory contents at both 2.7V and 4.5V to ensure that proper programming margins have been achieved. This ensures the PICmicro option over the voltage range of the system.

This final issue deals with the oscillator circuit on the application board. The voltage on MCLR/VPP must rise to the specified program mode entry voltage before the device executes any code. The crystal modes available on the PICmicro are not affected by this issue because the Oscillator Start-up Timer waits for 1024 oscillations before any code is executed. However, RC oscillators do not require any startup time and, therefore, the Oscillator Startup Timer is not used. The programmer must drive MCLR/VPP to the program mode entry voltage before the RC oscillator toggles four times. If the RC oscillator toggles four or more times, the program counter will be incremented to some value X. Now when the device enters programming mode, the program counter will not be zero and the programmer will start programming your code at an offset of X. There are several alternatives that can compensate for a slow rise rate on MCLR/VPP. The first method would be to not populate the R, program the device, and then insert the R. The other method would be to have the programming interface drive the OSC1 pin of the PICmicro to ground while programming. This will prevent any oscillations from occurring during programming.

Now all that is left is how to connect the application circuit to the programmer. This depends a lot on the programming environment and will be discussed in that section.

Programmer

The second consideration is the programmer. PIC16F8X MCUs only use serial programming and therefore all programmers supporting these devices will support ICSP. One issue with the programmer is the drive capability. As discussed before, it must be able to provide the specified rise rates on the ICSP signals and also provide enough current to power the application circuit. Appendix A shows an example driver board. This driver schematic does not show any buffer circuitry for RB6 and RB7. It is recommended that an evaluation be performed to determine if buffering is required. Another issue with the programmer is what VDD levels are used to verify the memory contents of the PICmicro. For instance, the PRO MATE II verifies program memory at the minimum and maximum VDD levels for the specified device and is therefore considered a production quality programmer. On the other hand, the PICSTART® Plus only verifies at 5V and is for prototyping use only. The Microchip programming specifications state that the program memory contents should be verified at both the minimum and maximum VDD levels that the application circuit will be operating. This implies that the application circuit must be able to handle the varying VDD voltages.

There are also several third party programmers that are available. You should select a programmer based on the features it has and how it fits into your programming environment. The *Microchip Development Systems Ordering Guide* (DS30177) provides detailed information on all our development tools. The *Microchip Third Party Guide* (DS00104) provides information on all of our third party tool developers. Please consult these two references when selecting a programmer. Many options exist including serial or parallel PC host connection, stand-alone operation, and single or gang programmers. Some of the third party developers include Advanced Transdata Corporation, BP Microsystems, Data I/O, Emulation Technology and Logical Devices.

Programming Environment

The programming environment will affect the type of programmer used, the programmer cable length, and the application circuit interface. Some programmers are well suited for a manual assembly line while others are desirable for an automated assembly line. You may want to choose a gang programmer to program multiple systems at a time.

The physical distance between the programmer and the application circuit affects the load capacitance on each of the programming signals. This will directly affect the drive strength needed to provide the correct signal rise rates and current. This programming cable must also be as short as possible and properly terminated and shielded or the programming signals may be corrupted by ringing or noise.

Finally, the application circuit interface to the programmer depends on the size constraints of the application circuit itself and the assembly line. A simple header can be used to interface the application circuit to the programmer. This might be more desirable for a manual assembly line where a technician plugs the programmer cable into the board. A different method is the use of spring loaded test pins (commonly referred to as pogo pins). The application circuit has pads on the board for each of the programming signals. Then there is a fixture that has pogo pins in the same configuration as the pads on the board. The application circuit or fixture is moved into position such that the pogo pins come into contact with the board. This method might be more suitable for an automated assembly line.

After taking into consideration the issues with the application circuit, the programmer, and the programming environment, anyone can build a high quality, reliable manufacturing line based on ICSP.

Other Benefits

ICSP provides other benefits, such as calibration and serialization. If program memory permits, it would be cheaper and more reliable to store calibration constants in program memory instead of using an external serial EEPROM. For example, your system has a thermistor which can vary from one system to another. Storing some calibration information in a table format allows the micrcontroller to compensate in software for external component tolerances. System cost can be reduced without affecting the required performance of the system by using software calibration techniques. But how does this relate to ICSP? The PICmicro has already been programmed with firmware that performs a calibration cycle. The calibration data is transferred to a calibration fixture. When all calibration data has been transferred, the fixture places the PICmicro in programming mode and programs the PICmicro with the calibration data. Application note *AN656, In-Circuit Serial Programming of Calibration Parameters Using a PICmicro Microcontroller*, shows exactly how to implement this type of calibration data programming.

The other benefit of ICSP is serialization. Each individual system can be programmed with a unique or random serial number. One such application of a unique serial number would be for security systems. A typical system might use DIP switches to set the serial number. Instead, this number can be burned into program memory thus reducing the overall system cost and lowering the risk of tampering.

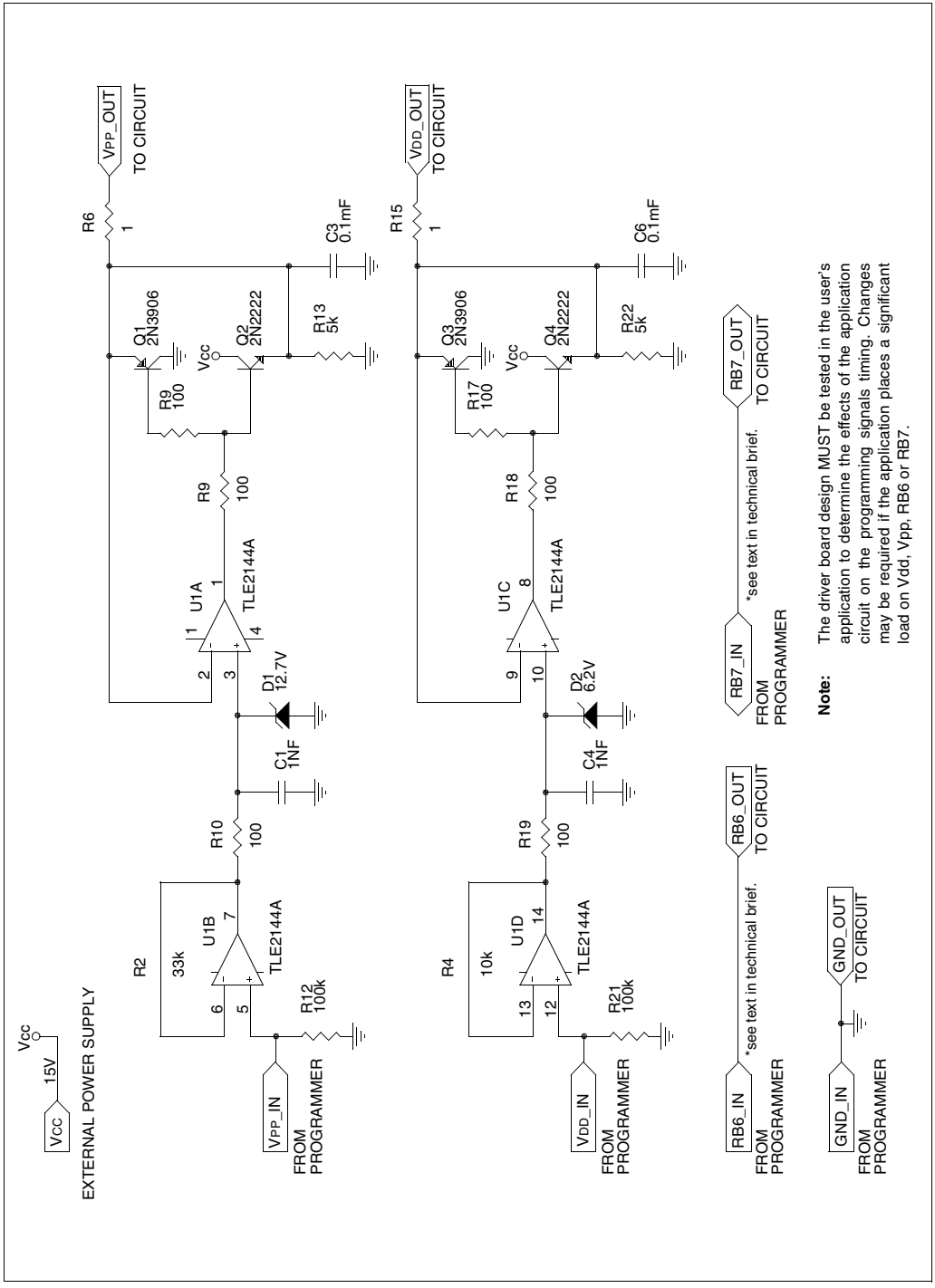
Field Programming of FLASH PICmicros

With the ISP interface circuitry already in place, these FLASH-based PICmicros can be easily reprogrammed in the field. These FLASH devices allow you to reprogram them even if they are code protected. A portable ISP programming station might consist of a laptop computer and programmer. The technician plugs the ISP interface cable into the application circuit and downloads the new firmware into the PICmicro. The next thing you know the system is up and running without those annoying “bugs”. Another instance would be that you want to add an additional feature to your system. All of your current inventory can be converted to the new firmware and field upgrades can be performed to bring your installed base of systems up to the latest revision of firmware.

CONCLUSION

Microchip Technology Inc. is committed to supporting your ICSP needs by providing you with our many years of experience and expertise in developing ICSP solutions. Anyone can create a reliable ICSP programming station by coupling our background with some forethought to the circuit design and programmer selection issues previously mentioned. Your local Microchip representative is available to answer any questions you have about the requirements for ICSP.

APPENDIX A: SAMPLE DRIVER BOARD SCHEMATIC



How to Implement ICSP™ Using PIC12C5XX OTP MCUs

Author: *Thomas Schmidt*
Microchip Technology Inc.

INTRODUCTION

The technical brief describes how to implement in-circuit serial programming (ICSP™) using the PIC12C5XX OTP PICmicro® MCU.

ICSP is a simple way to manufacture your board with an unprogrammed PICmicro and program the device just before shipping the product. Programming the PIC12C5XX MCU in-circuit has many advantages for developing and manufacturing your product.

- **Reduces inventory of products with old firmware**. With ICSP, the user can manufacture product without programming the PICmicro MCU. The PICmicro will be programmed just before the product is shipped.
- **ICSP in production**. New software revisions or additional software modules can be programmed during production into the PIC12C5XX MCU.
- **ICSP in the field.** Even after your product has been sold, a service man can update your program with new program modules.
- **One hardware with different software**. ICSP allows the user to have one hardware, whereas the PIC12C5XX MCU can be programmed with different types of software.
- **Last minute programming**. Last minute programming can also facilitate quick turnarounds on custom orders for your products.

IN-CIRCUIT SERIAL PROGRAMMING

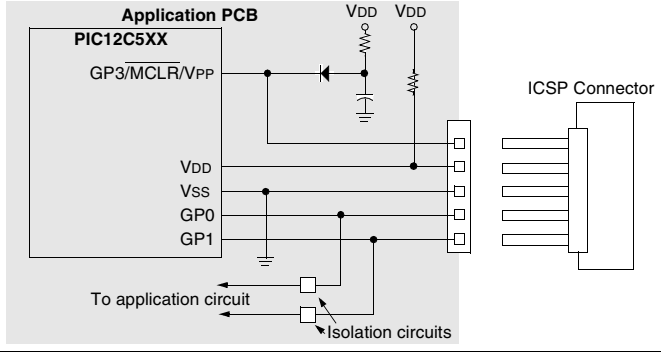
To implement ICSP into an application, the user needs to consider three main components of an ICSP system: Application Circuit, Programmer and Programming Environment.

Application Circuit

During the initial design phase of the application circuit, certain considerations have to be taken into account. Figure 1 shows and typical circuit that addresses the details to be considered during design. In order to implement ICSP on your application board you have to put the following issues into consideration:

1. Isolation of the GP3/$\overline{\text{MCLR}}$/V_{PP} pin from the rest of the circuit.
2. Isolation of pins GP1 and GP0 from the rest of the circuit.
3. Capacitance on each of the V_{DD}, GP3/$\overline{\text{MCLR}}$/V_{PP}, GP1, and GP0 pins.
4. Interface to the programmer.
5. Minimum and maximum operating voltage for V_{DD}.

FIGURE 1: TYPICAL APPLICATION CIRCUIT



PRO MATE, PICSTART and PICmicro are registered trademarks and ICSP is a trademark of Microchip Technology Inc.

Isolation of the GP3/MCLR/VPP Pin from the Rest of the Circuit

PIC12C5XX devices have two ways of configuring the MCLR pin:

- MCLR can be connected either to an external RC circuit or
- MCLR is tied internally to VDD

When GP3/MCLR/VPP pin is connected to an external RC circuit, the pull-up resistor is tied to VDD, and a capacitor is tied to ground. This circuit can affect the operation of ICSP depending on the size of the capacitor.

Another point of consideration with the GP3/MCLR/VPP pin, is that when the PICmicro is programmed, this pin is driven up to 13V and also to ground. Therefore, the application circuit must be isolated from the voltage coming from the programmer.

When MCLR is tied internally to VDD, the user has only to consider that up to 13V are present during programming of the GP3/MCLR/VPP pin. This might affect other components connected to that pin.

For more information about configuring the GP3/MCLR/VPP internally to VDD, please refer to the PIC12C5XX data sheet (DS40139).

Isolation of Pins GP1 and GP0 from the Rest of the Circuit

Pins GP1 and GP0 are used by the PICmicro for serial programming. GP1 is the clock line and GP0 is the data line.

GP1 is driven by the programmer. GP0 is a bi-directional pin that is driven by the programmer when programming and driven by the PICmicro when verifying. These pins must be isolated from the rest of the application circuit so as not to affect the signals during programming. You must take into consideration the output impedance of the programmer when isolating GP1 and GP0 from the rest of the circuit. This isolation circuit must account for GP1 being an input on the PICmicro and for GP0 being bi-directional pin.

For example, PRO MATE® II has an output impedance of 1 kΩ. If the design permits, these pins should not be used by the application. This is not the case with most designs. As a designer, you must consider what type of circuitry is connected to GP1 and GP0 and then make a decision on how to isolate these pins.

Total Capacitance on VDD, GP3/MCLR/VPP, GP1, and GP0

The total capacitance on the programming pins affects the rise rates of these signals as they are driven out of the programmer. Typical circuits use several hundred microfarads of capacitance on VDD, which helps to dampen noise and improve electromagnetic interference. However, this capacitance requires a fairly strong driver in the programmer to meet the rise rate timings for VDD.

Interface to the Programmer

Most programmers are designed to simply program the PICmicro itself and don't have strong enough drivers to power the application circuit.

One solution is to use a driver board between the programmer and the application circuit. The driver board needs a separate power supply that is capable of driving the VPP, VDD, GP1, and GP0 pins with the correct ramp rates and also should provide enough current to power-up the application circuit.

The cable length between the programmer and the circuit is also an important factor for ICSP. If the cable between the programmer and the circuit is too long, signal reflections may occur. These reflections can momentarily cause up to twice the voltage at the end of the cable, that was sent from the programmer. This voltage can cause a latch-up. In this case, a termination resistor has to be used at the end of the signal line.

Minimum and Maximum Operating Voltage for VDD

The PIC12C5XX programming specification states that the device should be programmed at 5V. Special considerations must be made if your application circuit operates at 3V only. These considerations may include totally isolating the PICmicro during programming. The other point of consideration is that the device must be verified at minimum and maximum operation voltage of the circuit in order to ensure proper programming margin.

For example, a battery driven system may operate from three 1.5V cells giving an operating voltage range of 2.7V to 4.5V. The programmer must program the device at 5V and must verify the program memory contents at both 2.7V and 4.5V to ensure that proper programming margins have been achieved.

THE PROGRAMMER

PIC12C5XX MCUs only use serial programming and, therefore, all programmers supporting these devices will support the ICSP. One issue with the programmer is the drive capability. As discussed before, it must be able to provide the specified rise rates on the ICSP signals and also provide enough current to power the application circuit. It is recommended that you buffer the programming signals.

Another point of consideration for the programmer is what VDD levels are used to verify the memory contents of the PICmicro. For instance, the PRO MATE II verifies program memory at the minimum and maximum VDD levels for the specified device and is therefore considered a production quality programmer. On the other hand, the PICSTART® Plus only verifies at 5V and is for prototyping use only. The PIC12C5XX programming specifications state that the program memory contents should be verified at both the minimum and maximum VDD levels that the application circuit will be operating. This implies that the application circuit must be able to handle the varying VDD voltages.

There are also several third-party programmers that are available. You should select a programmer based on the features it has and how it fits into your programming environment. The *Microchip Development Systems Ordering Guide* (DS30177) provides detailed information on all our development tools. The *Microchip Third Party Guide* (DS00104) provides information on all of our third party development tool developers. Please consult these two references when selecting a programmer. Many options exist including serial or parallel PC host connection, stand-alone operation, and single or gang programmers.

PROGRAMMING ENVIRONMENT

The programming environment will affect the type of programmer used, the programmer cable length, and the application circuit interface. Some programmers are well suited for a manual assembly line while others are desirable for an automated assembly line. A gang programmer should be chosen for programming multiple MCUs at one time. The physical distance between the programmer and the application circuit affects the load capacitance on each of the programming signals. This will directly affect the drive strength needed to provide the correct signal rise rates and current. Finally, the application circuit interface to the programmer depends on the size constraints of the application circuit itself and the assembly line. A simple header can be used to interface the application circuit to the programmer. This might be more desirable for a manual assembly line where a technician plugs the programmer cable into the board.

A different method is the uses spring loaded test pins (often referred as pogo-pins). The application circuit has pads on the board for each of the programming signals. Then there is a movable fixture that has pogo pins in the same configuration as the pads on the board. The application circuit is moved into position and the fixture is moved such that the spring loaded test pins come into contact with the board. This method might be more suitable for an automated assembly line.

After taking into consideration the issues with the application circuit, the programmer, and the programming environment, anyone can build a high quality, reliable manufacturing line based on ICSP.

OTHER BENEFITS

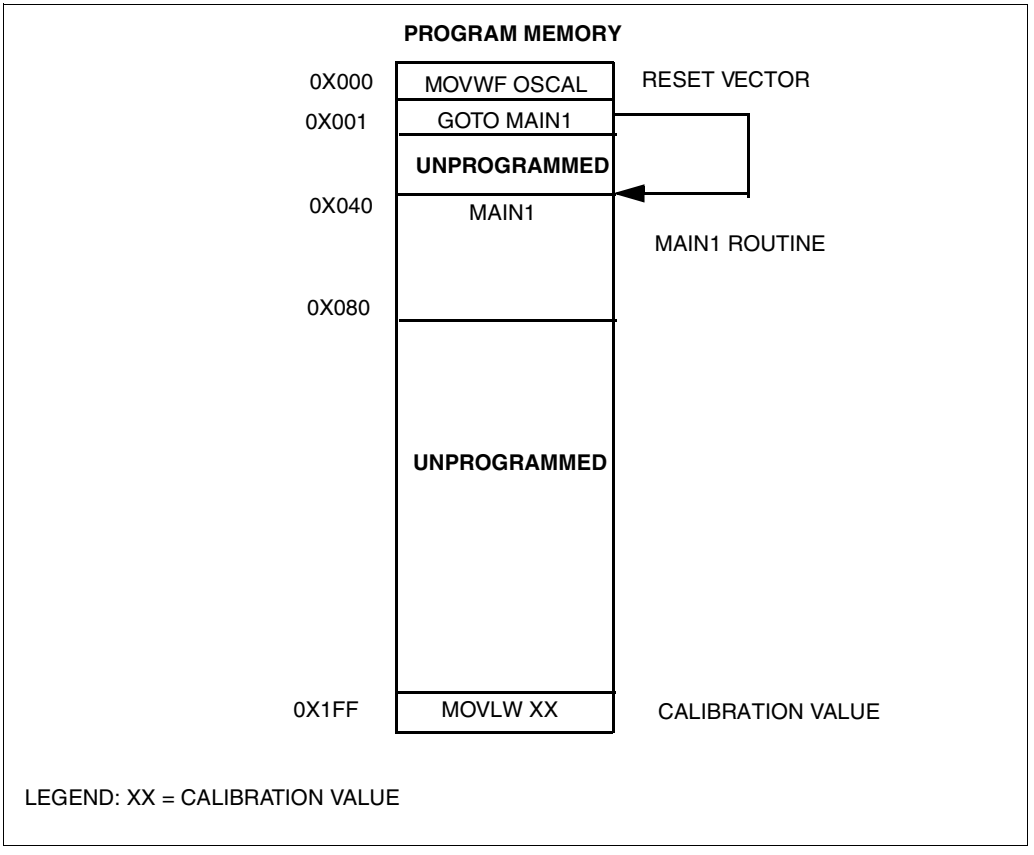
ICSP provides several other benefits such as calibration and serialization. If program memory permits, it would be cheaper and more reliable to store calibration constants in program memory instead of using an external serial EEPROM.

Field Programming of PICmicro OTP MCUs

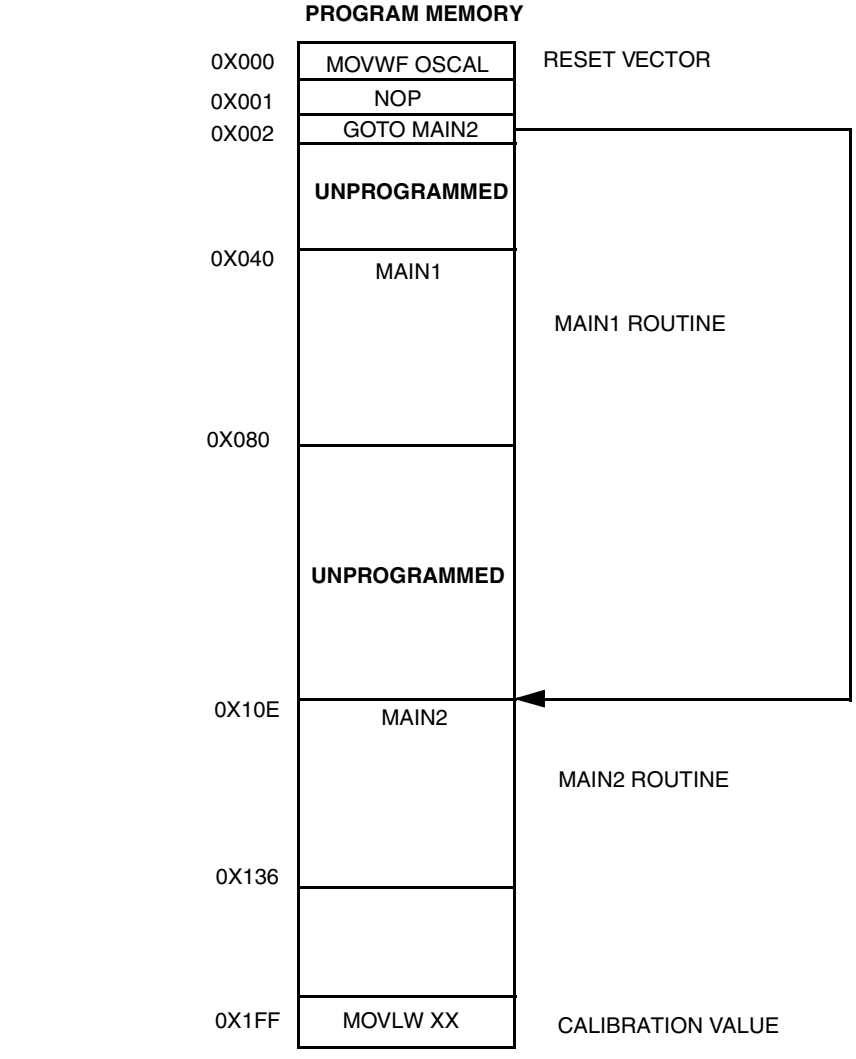
An OTP device is not normally capable of being reprogrammed, but the PICmicro architecture gives you this flexibility provided the size of your firmware is less than half that of the desired device.

This method involves using jump tables for the reset and interrupt vectors. Example 1 shows the location of a main routine and the reset vector for the first time a device with 0.5K-words of program memory is programmed. Example 2 shows the location of a second main routine and its reset vector for the second time the same device is programmed. You will notice that the `GOTO Main` that was previously at location 0x0002 is replaced with an NOP. An NOP is a program memory location with all the bits programmed as 0s. When the reset vector is executed, it will execute an NOP and then a `GOTO Main1` instruction to the new code.

EXAMPLE 1: LOCATION OF THE FIRST MAIN ROUTINE AND ITS INTERRUPT VECTOR



EXAMPLE 2: LOCATION OF THE SECOND MAIN ROUTINE AND IT INTERRUPT VECTOR (AFTER SECOND PROGRAMMING)



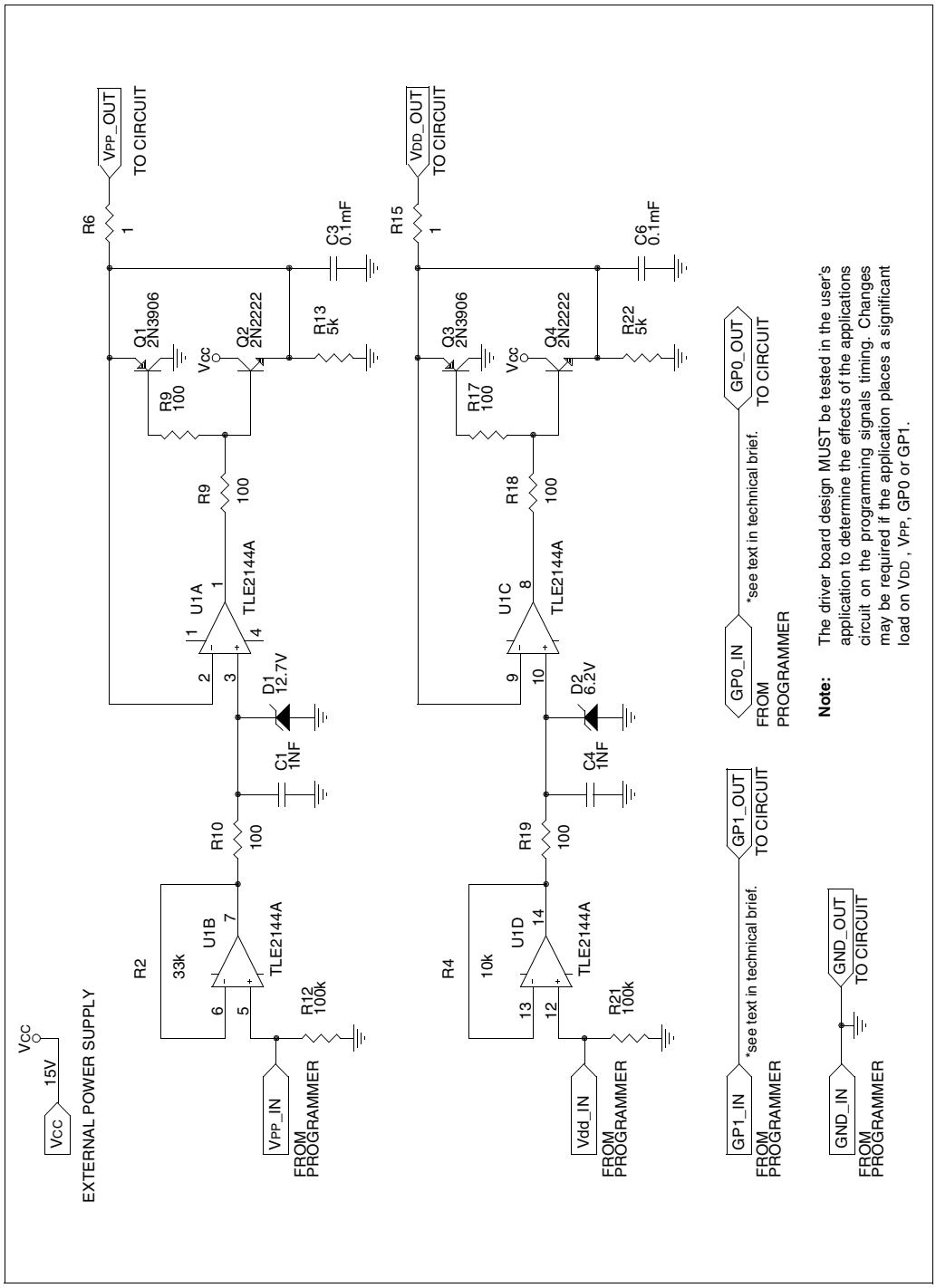
LEGEND: XX = CALIBRATION VALUE

Since the program memory of the PIC12C5XX devices is organized in 256 x 12 word pages, placement of such information as look-up tables and CALL instructions must be taken into account. For further information, please refer to application note *AN581, Implementing Long Calls* and application note *AN556, Implementing a Table Read*.

CONCLUSION

Microchip Technology Inc. is committed to supporting your ICSP needs by providing you with our many years of experience and expertise in developing in-circuit system programming solutions. Anyone can create a reliable in-circuit system programming station by coupling our background with some forethought to the circuit design and programmer selection issues previously mentioned. Your local Microchip representative is available to answer any questions you have about the requirements for ICSP.

APPENDIX A: SAMPLE DRIVER BOARD SCHEMATIC



NOTES:

TB020

PIC12C67X Emulation Using PIC16C72 PICMASTER® Emulator Probe

Author: *Rob Stein*
Microchip Technology Inc.

INTRODUCTION

This technical brief describes how to use the PIC16C72 PICMASTER® emulator probe for PIC12C67X emulation.

OVERVIEW

Many simple PIC12C67X applications can be developed using the MPLAB™ simulator (Version 3.31 or higher), for complex applications real-time emulation may be required. Since the PIC16C72 shares the same upward expanded memory map and similar pin functions as the PIC12C67X, the PICMASTER emulator probe 16J (AC165009) can be used to emulate most PIC12C67X functions. The PIC16C710, PIC16C711 or PIC16C715 emulator probes are not recommended for PIC12C67X emulation due to only 5 bits in the PORTA (address 05 hex) register map versus the 6 bits for the PIC12C67X.

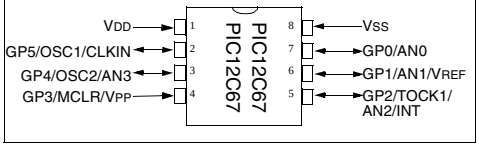
A custom bond-out chip is being designed specifically for emulation of the PIC12C67X (scheduled for completion in Q3 1998) and this will eliminate the need for this technical brief.

HARDWARE EMULATION RECOMMENDATIONS

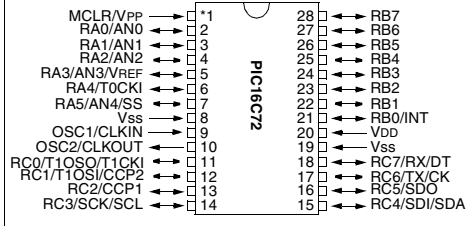
Your target PCB will accept the 8 pin PDIP or SOIC pin out of the PIC12C67X. An adapter socket must be constructed to interface from your 8 pin target to the 28 pin DIP socket on the emulator probe.

PACKAGE TYPES

PDIP, SOIC



SDIP, SOIC, Windowed Side Brazed Ceramic



Building the adapter socket

Many PIC12C67X functions are multiplexed into a single pin. For example, the GP2/T0CK1/AN2/INT pin, can be configured as a digital I/O, Timer 0 counter input, A/D input, or external interrupt pin. This highly multiplexed PIC12C67X pin does not exactly match a corresponding PIC16C72 pin, however most applications will use this pin in only one of it's four configurations. Therefore, a single PIC12C67X pin function can be mapped into the corresponding pin on PORTA or PORTB of the PIC16C72.

If any A/D channels are enabled then it is necessary to map all PIC12C67X GPIO (Digital I/O's) to PORTB of the PIC16C72. Using PORTB will enable emulation of the external interrupt, programmable pull-up resistors, and will simplify changes to the ADCON1 register.

Your adapter socket pin out must be customized for your specific application. The following tables show the pin out for six common applications.

TABLE 1: 6 DIGITAL I/O'S

PIC12C67X Pin Function		Equivalent PIC16C72 Function	
Name	**#**	**Name**	**#**
VDD	1	VDD	20
GP5/OSC1/CLKIN	2	RA5/AN4	7
GP4/OSC2/AN3/CLKOUT	3	RA4/TOCK1	6
GP3/MCLR/Vpp	4	RA3/AN3/Vref	5
GP2/TOCK1/AN2/INT	5	RA2/AN2	4
GP1/AN1/Vref	6	RA1/AN1	3
GP0/AN0	7	RA0/AN0	2
VSS	8	VSS	8,19

Note 1: If you plan to use the GPIO pull-up resistors then map to PORTB.

TABLE 2: 4 ANALOG AND 2 DIGITAL I/O'S

PIC12C67X Pin Function		Equivalent PIC16C72 Function	
Name	**#**	**Name**	**#**
VDD	1	VDD	20
GP5/OSC1/CLKIN	2	RB5	26
GP4/OSC2/AN3/CLKOUT	3	RA3/AN3/Vref	5
GP3/MCLR/Vpp	4	RB3	24
GP2/TOCK1/AN2/INT	5	RA2/AN2	4
GP1/AN1/Vref	6	RA1/AN1	3
GP0/AN0	7	RA0/AN0	2
VSS	8	VSS	8,19

TABLE 3: 3 ANALOG AND 3 DIGITAL I/O'S

PIC12C67X Pin Function		Equivalent PIC16C72 Function	
Name	**#**	**Name**	**#**
VDD	1	VDD	20
GP5/OSC1/CLKIN	2	RB5	26
GP4/OSC2/AN3/CLKOUT	3	RB4	25
GP3/MCLR/Vpp	4	RB3	24
GP2/TOCK1/AN2/INT	5	RA3/AN3/Vref	5
GP1/AN1/Vref	6	RA1/AN1	3
GP0/AN0	7	RA0/AN0	2
VSS	8	VSS	8,19

TABLE 4: 2 ANALOG AND 4 DIGITAL I/O'S

PIC12C67X Pin Function		Equivalent PIC16C72 Function	
Name	**#**	**Name**	**#**
VDD	1	VDD	20
GP5/OSC1/CLKIN	2	RB5	26
GP4/OSC2/AN3/CLKOUT	3	RB4	25
GP3/MCLR/Vpp	4	RB3	24
GP2/TOCK1/AN2/INT	5	RB2	23
GP1/AN1/Vref	6	RA1/AN1	3
GP0/AN0	7	RA0/AN0	2
VSS	8	VSS	8,19

TABLE 5: 1 ANALOG AND 5 DIGITAL I/O'S

PIC12C67X Pin Function		Equivalent PIC16C72 Function	
Name	**#**	**Name**	**#**
VDD	1	VDD	20
GP5/OSC1/CLKIN	2	RB5	26
GP4/OSC2/AN3/CLKOUT	3	RB4	25
GP3/MCLR/Vpp	4	RB3	24
GP2/TOCK1/AN2/INT	5	RB2	23
GP1/AN1/Vref	6	RB1	22
GP0/AN0	7	RA0/AN0	2
VSS	8	VSS	8,19

Note 1: If you plan to use the GPIO pull-up resistors then map to PORTB.

TABLE 6: 2 ANALOG, 3 DIGITAL I/O'S AND 1 EDGE TRIGGERED INTERRUPT

PIC12C67X Pin Function		Equivalent PIC16C72 Function	
Name	**#**	**Name**	**#**
VDD	1	VDD	20
GP5/OSC1/CLKIN	2	RB5	26
GP4/OSC2/AN3/CLKOUT	3	RB4	25
GP3/MCLR/Vpp	4	RB3	24
GP2/TOCK1/AN2/INT	5	RB0/INT	21
GP1/AN1/Vref	6	RA1/AN1	3
GP0/AN0	7	RA0/AN0	2
VSS	8	VSS	8,19

HARDWARE DIFFERENCES AND WORK-AROUNDS

The PIC12C67X has more flexibility in selecting A/D versus digital I/O pins (see table 2, ADCON1 register). To work around this difference, enable more A/D channels on PORTA of the PIC16C72 and map all digital I/O's to PORTB of the PIC16C72 (and ignore the extra 16C72 A/D channels).

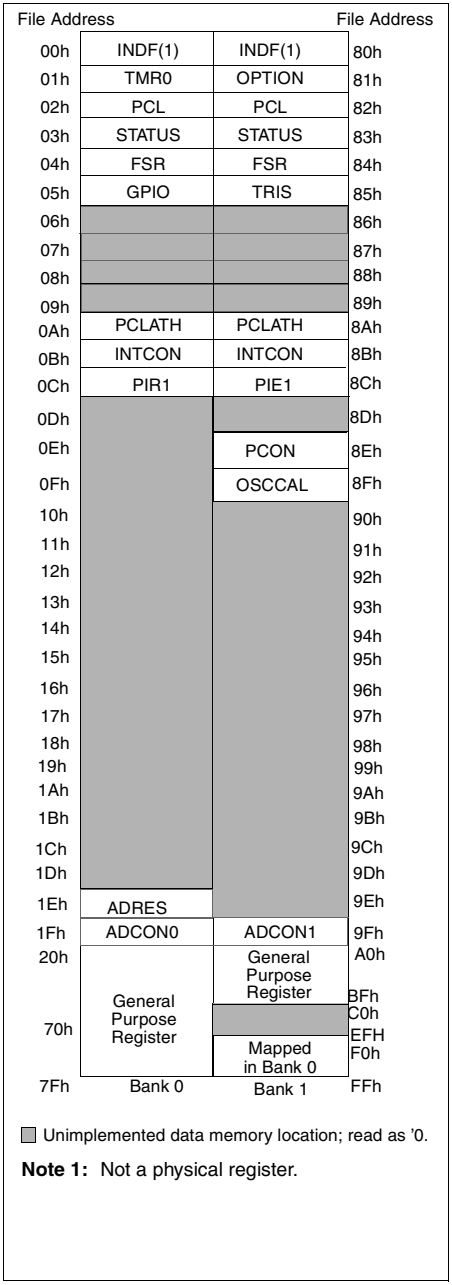
The PIC12C67X also has an on chip oscillator with the ability to output it's clock (to CLKOUT pin). The internal oscillator option can be emulated with the PIC16C72 by using a 4Mhz 'canned' clock oscillator, with the probe jumper set to "INT CLK".

The RA4/T0CK1 pin of the PIC16C72 when configured as an output is open drain, and therefore will need a pull-up resistor to emulate the GP4 PIC12C67X pin output.

SOFTWARE EMULATION RECOMMENDATIONS

The PIC12C67X and PIC16C72 both have 2K words of program memory and 128 bytes of RAM. As is indicated by the following register file map, the PIC12C67X is a sub-set of the PIC16C72. The 'extra' registers of the PIC16C72 can be ignored while emulating.

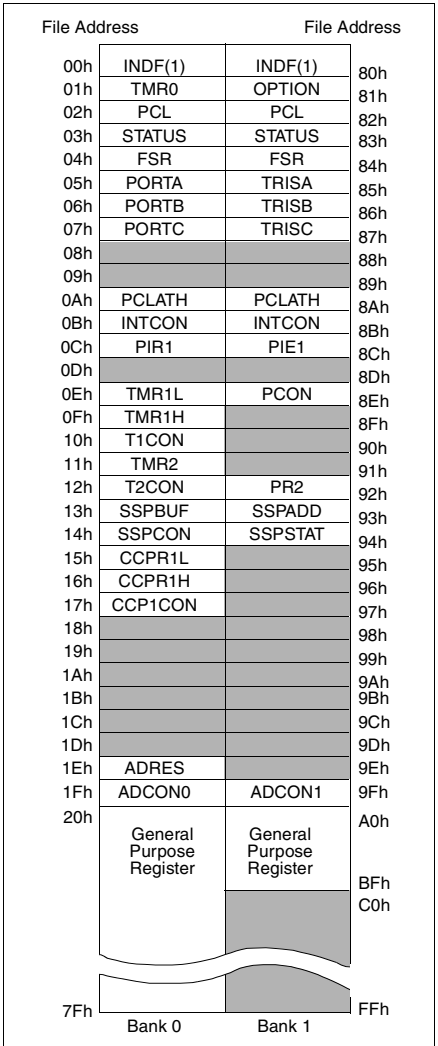
FIGURE 1: PIC12C67X REGISTER FILE MAP



Unimplemented data memory location; read as '0.

Note 1: Not a physical register.

FIGURE 2: PIC16C72 REGISTER FILE MAP



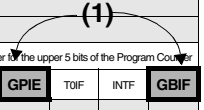
Unimplemented data memory location; read as '0.

Note 1: Not a physical register.

SPECIAL FUNCTION REGISTER SUMMARY (BANK 0)

PIC12C67X

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
					Bank 0				
00h(1)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)							
01h	TMR0	Timer0 module's register							
02h(1)	PCL	Program Counter's (PC) Least Significant Byte							
03h(1)	STATUS	IRP(4)	RP1(4)	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
04h(1)	FSR	Indirect data memory address pointer							
05h	GPIO	—	—	GP5	GP4	GP3	GP2	GP1	GP0
06h	—	Unimplemented							
07h	—	Unimplemented							
08h	—	Unimplemented							
09h	—	Unimplemented							
0Ah(1,2)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter				
0Bh(1)	INTCON	GIE	PEIE	T0IE	INTE	**GPIE**	T0IF	INTF	**GBIF**
0Ch	PIR1	—	ADIF	—	—	—	—	—	—
0Dh	—	Unimplemented							
0Eh	—	Unimplemented							
0Fh	—	Unimplemented							
10h	—	Unimplemented							
11h	—	Unimplemented							
12h	—	Unimplemented							
13h	—	Unimplemented							
14h	—	Unimplemented							
15h	—	Unimplemented							
16h	—	Unimplemented							
17h	—	Unimplemented							
18h	—	Unimplemented							
19h	—	Unimplemented							
1Ah	—	Unimplemented							
1Bh	—	Unimplemented							
1Ch	—	Unimplemented							
1Dh	—	Unimplemented							
1Eh	ADRES	A/D Result Register							
1Fh	ADCON0	ADCS1	ADCS0	r	CHS1	CHS0	GO/$\overline{DONE}$	r	ADON



1: The PIC12C67X bit names are different than the corresponding PIC16C72 names, but the functions are the same.

PIC16C72

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)							
TMR0	Timer0 module's register							
PCL	Program Counter's (PC) Least Significant Byte							
STATUS	IRP(4)	RP1(4)	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
FSR	Indirect data memory address pointer							
PORTA	—	—	PORTA Data Latch when written: PORTA pins when read					
PORTB	PORTB Data Latch when written: PORTB pins when read							
PORTC	PORTC Data Latch when written: PORTC pins when read							
—	Unimplemented							
—	Unimplemented							
PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter				
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
PIR1	—	ADIF	—	—	SSPIF	CCP1IF	TMR2IF	TMR1IF
—	Unimplemented							
TMR1L	Holding register for the Least Significant Byte of the 16-bit TMR1 register							
TMR1H	Holding register for the Most Significant Byte of the 16-bit TMR1 register							
T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON
TMR2	Timer2 module's register							
T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register							
SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
CCPR1L	Capture/Compare/PWM Register (LSB)							
CCPR1H	Capture/Compare/PWM Register (MSB)							
CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0
—	Unimplemented							
—	Unimplemented							
—	Unimplemented							
—	Unimplemented							
—	Unimplemented							
—	Unimplemented							
ADRES	A/D Result Register							
ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	—	ADON

SPECIAL FUNCTION REGISTER SUMMARY (BANK 1)

PIC12C67X

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		Bank 1							
80h(1)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)							
81h	OPTION	GPPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
82h(1)	PCL	Program Counter's (PC) Least Significant Byte							
83h(1)	STATUS	IRP(4)	RP1(4)	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
84h(1)	FSR	Indirect data memory address pointer							
85h	TRIS	—	—	GPIO Data Direction Register					
86h	—	Unimplemented							
87h	—	Unimplemented							
88h	—	Unimplemented							
89h	—	Unimplemented							
8Ah(1,2)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the PC				
8Bh(1)	INTCON	GIE	PEIE	T0IE	INTE	GPIE	T0IF	INTF	GPIF
8Ch	PIE1	—	ADIE	—	—	—	—	—	—
8Dh	—	Unimplemented							
8Eh	PCON	—	—	—	—	—	—	$\overline{POR}$	—
8Fh	OSCCAL	CAL3	CAL2	CAL1	CAL0	CALFST	CALSLW	—	—
90h	—	Unimplemented							
91h	—	Unimplemented							
92h	—	Unimplemented							
93h	—	Unimplemented							
94h	—	Unimplemented							
95h	—	Unimplemented							
96h	—	Unimplemented							
97h	—	Unimplemented							
98h	—	Unimplemented							
99h	—	Unimplemented							
9Ah	—	Unimplemented							
9Bh	—	Unimplemented							
9Ch	—	Unimplemented							
9Dh	—	Unimplemented							
9Eh	—	Unimplemented							
9Fh	ADCON1	—	—	—	—	—	PCFG2	PCFG1	PCFG0

Note 1: The PIC12C67X bit names are different than the corresponding PIC16C72 names, but the functions are the same.

2: The OSCCAL register is unimplemented in the PIC16C72.

PIC16C72

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)							
OPTION	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
PCL	Program Counter's (PC) Least Significant Byte							
STATUS	IRP(4)	RP1(4)	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
FSR	Indirect data memory address pointer							
TRISA	—	—	PORTA Data Direction Register					
TRISB	PORTB Data Direction Register							
TRISC	PORTC Data Direction Register							
—	Unimplemented							
—	Unimplemented							
PCLATH	—	—	—	Write Buffer for the upper 5 bits of the PC				
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
PIE1	—	ADIE	—	—	SSPIE	CCP1IE	TMR2IE	TMR1IE
—	Unimplemented							
PCON	—	—	—	—	—	—	$\overline{POR}$	$\overline{BOR}$
—	Unimplemented							
—	Unimplemented							
—	Unimplemented							
PR2	Timer2 Period Register							
SSPADD	Synchronous Serial Port (I^2C mode) Address Register							
SSPSTAT	—	—	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
—	Unimplemented							
—	Unimplemented							
—	Unimplemented							
—	Unimplemented							
—	Unimplemented							
—	Unimplemented							
—	Unimplemented							
—	Unimplemented							
—	Unimplemented							
—	Unimplemented							
ADCON1	—	—	—	—	—	PCFG2	PCFG1	PCFG0

Your application software can be written with the MPASM conditional assembly feature. When emulating, assemble for the PIC16C72, when programming, re-assemble for the PIC12C67X. The attached source code (AtoD.asm) has been written to show a conditional assembly example and re-mapping of the registers.

SOFTWARE EMULATION DIFFERENCES AND WORK AROUNDS:

Both devices have an ADCON1 register with 3 bits of control (PCFG0, 1, &2), however the PIC12C67X has much finer control over it's individual A/D pins. Therefore, if only one PIC12C67X A/D channel is needed (ADCON1 = xxxx x110), the PIC16C72 emulator will be configured as three A/D channels enabled (ADCON1= xxxx x100). Only one channel will be connected from the emulator to the PIC12C67X and the other two channels will be ignored.

The ADCON1 register differences are:

TABLE 2A: PIC16C72 PCFG2:PCFG0 A/D Control bits

PCFG2 : PCFG0	RA5	RA3	RA2	RA1	RA0	VREF
000	A	A	A	A	A	VDD
001	A	VREF	A	A	A	RA3
010	A	A	A	A	A	VDD
011	A	VREF	A	A	A	RA3
100	D	A	D	A	A	VDD
101	D	VREF	D	A	A	RA3
110	D	D	D	D	D	-----
111	D	D	D	D	D	-----

TABLE 2B: PIC12C67X PCFG2:PCFG0 A/D Control bits

PCFG2 : PCFG0	GP4	GP2	GP1	GP0	VREF
000	A	A	A	A	VDD
001	A	A	Vref	A	GP1
010	D	A	A	A	VDD
011	D	A	Vref	A	GP1
100	D	D	A	A	VDD
101	D	D	Vref	A	GP1
110	D	D	D	A	VDD
111	D	D	D	D	-----

Downloading HEX Files to External FLASH Memory Using PIC17CXXX PICmicro® Microcontrollers

Author: *Rodger Richey*
Microchip Technology Inc.

INTRODUCTION

The PIC17CXXX devices have the capability to interface external FLASH memory into the 64K x 16 program memory space. Coupled with this feature is the ability to read and write to the entire program memory of the device. Using one of the standard serial interfaces on the PICmicro (USART, SPI, I^2C™), a complete hex file can be downloaded into the external FLASH memory by a bootloader program. The PIC17CXXX family consists of seven devices as shown in Table 1.

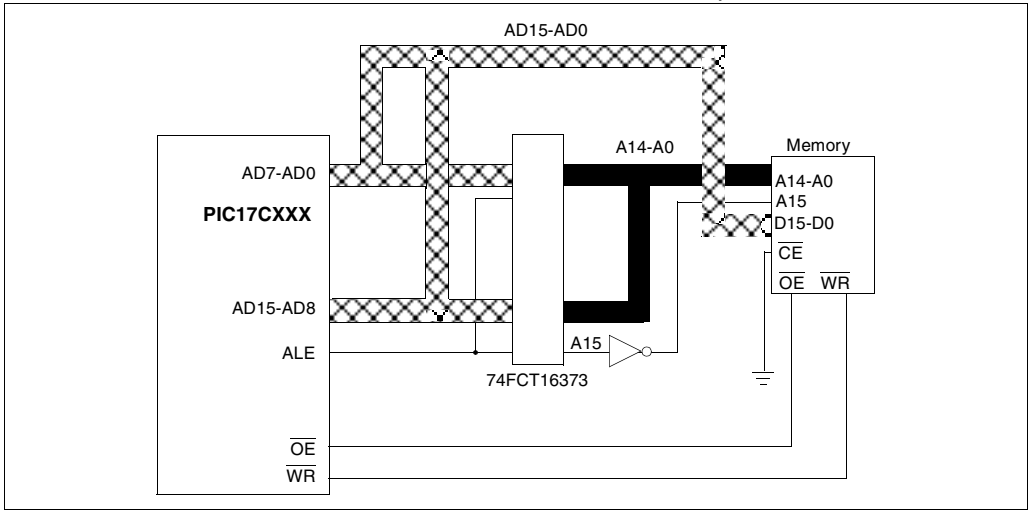
TABLE 1: FEATURES LIST

Features	PIC17C42A	PIC17C43	PIC17C44	PIC17C756A	PIC17C762	PIC17C766
Max Freq for Ops	33 MHz	33 MHz	33 MHz	33 MHz	33 MHz	33 MHz
Op Voltage Range	2.5V - 6.0V	2.5V - 6.0V	2.5V - 6.0V	3.0V - 5.5V	3.0V - 5.5V	3.0V - 5.5V
Prog Memory x16	2K	4K	8K	16K	8K	16K
Data Memory (bytes)	232	454	454	902	678	902
Hardware Multiplier	Yes	Yes	Yes	Yes	Yes	Yes
Timers	4	4	4	4	4	4
Capture Inputs	2	2	2	4	4	4
PWM Outputs	2	2	2	3	3	3
USART/SCI	1	1	1	2	2	2
A/D Channels	-	-	-	12	16	16
Power-on Reset	Yes	Yes	Yes	Yes	Yes	Yes
Brown-out Reset	-	-	-	Yes	Yes	Yes
ICSP	-	-	-	Yes	Yes	Yes
Watchdog Timer	Yes	Yes	Yes	Yes	Yes	Yes
Interrupt Sources	11	11	11	18	18	18
I/O pins	33	33	33	50	66	66

FLASH SELECTION

The first decision is what FLASH memory to use in the circuit. This document will focus on the Am29F100 from AMD. This device has a selectable memory/interface size: 128K x 8 or 64K x 16. The 16-bit interface is chosen because the PIC17CXXX devices have 16-bit wide program memory. The address line A15 may need to be inverted depending on the PICmicro internal OTP memory size and the FLASH memory selected. The AMD device needs to access address locations 2AAAh and 5555h for program and erase operations. For PICmicro microcontrollers with 8K or less program memory, no inversion is necessary, but is required for the 16K and larger devices. The address location 2AAAh in the FLASH memory is mapped on top of internal program memory, which takes precedence. Any access to 2AAAh will be to the internal OTP memory and not to the external FLASH memory. The inversion is transparent to the designer except that program or erase operations will use address locations AAAAh and D555h instead due to the inversion. The Technical Brief (TB027), Simplifying External Memory Connections of PIC17CXXX PICmicro mirocontrollers, covers the memory mapping and circuit connection considerations in more detail. Figure 1 shows a block diagram for the external memory connections.

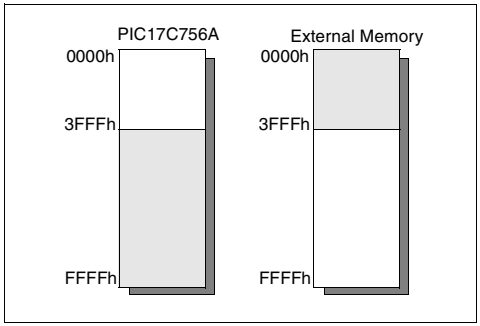
FIGURE 1: EXTERNAL MEMORY INTERFACE BLOCK DIAGRAM (x16 DEVICES



MICROCONTROLLER CONFIGURATION

The microcontroller has several operating modes. The first being **Microcontroller** mode, which uses only the internal OTP program memory. In this mode all I/O pins function as I/O pins. The second mode is **Microprocessor** mode, which uses only external memory. In this mode, 19 of the I/O pins function as the external memory interface (3 for control, 16 for address/data). The final mode is **Extended Microcontroller** mode, which uses internal OTP program memory. The remainder of 64K is external to the device. This mode must be used to program the external FLASH memory and the bootloader routine must reside in the OTP memory. Refer to the PIC17C7XX data sheet (DS30289) or the PIC17C4X data sheet (DS30412) for more information about processor modes. Figure 2 shows the memory map configuration for extended microcontroller mode for the PIC17C756A.

FIGURE 2: PIC17C756A IN EXTENDED MICROCONTROLLER MODE



HEX FILE FORMAT

The HEX file to be programmed into program memory will be read into the microcontroller using one of its standard interface modules: USART, SPI, or I^2C. The formats supported by the Microchip development tools are the Intel Hex Format (INHX8M), Intel Split Hex Format (INHX8S), and the Intel Hex 32 Format (INHX32). The format required by the PIC17CXXX devices is the INHX32 due to the 64K of address space. Please refer to Appendix A in the MPASM User's Guide (DS33014) for more information about HEX file formats. The INHX32 format supports 32-bit addresses using a linear address record. The basic format of the INHX32 hex file is:

```
:BBAAAATTHHHH...HHHHCC
```

Each data record begins with a 9 character prefix and always ends with a 2 character checksum. All records begin with a '**:**' regardless of the format. The individual elements are described below.

- **BB** - is a two digit hexadecimal byte count representing the number of data bytes that will appear on the line. Divide this number by two to get the number of words per line.
- **AAAA** - is a four digit hexadecimal address representing the starting address of the data record. Format is high byte first followed by low byte. The address is doubled because this format only supports 8-bits (to find the real PICmicro address, simply divide the value **AAAA** by 2).
- **TT** - is a two digit record type that will be '00' for data records, '01' for end of file records and '04' for extended address record.
- **HHHH** - is a four digit hexadecimal data word. Format is low byte followed by high byte. There will be **BB**/2 data words following **TT.**
- CC - is a two digit hexadecimal checksum that is the two's complement of the sum of all the preceding bytes in the line record.

The HEX file is composed of ASCII characters 0 thorough 9 and A to F and the end of each line has a carriage return and linefeed. The downloader code in the PICmicro must convert the ASCII characters to binary numbers for use in programming.

PICmicro CODE

The code for the PIC17CXXX devices was written using the MPLAB-C17 C compiler. A demo version of the MPLAB-C17 C compiler is available off the Microchip website, *www.microchip.com*. This code uses USART2 on the PIC17C756A as the interface to the PC. In addition to USART2, two I/O pins are used to implement hardware handshaking with the PC host. Handshaking must be used because the program time of the FLASH memory prevents the PC from simply streaming the data down to the PICmicro microcontroller. The PICmicro microcontroller itself does not have enough RAM to buffer the incoming data while the FLASH is programming. Listing 1 shows the C code. Figure 3 shows a flowchart for the downloader code.

In this particular example, the hardware USART2 is used to download hex files from the PC host. Hardware handshaking is used to communicate with the PC. The function `DataRdyU2` properly asserts the handshake signals to the PC to receive one byte of data.

Two other functions not listed read in a byte (`Hex8in`) or a word (`Hex16in`) and return the binary value of the ASCII characters read. `Hex8in` reads two characters and converts them to an 8-bit value. `Hex16in` reads in 4 characters and converts them to binary. The format for `Hex16in` is high byte then low byte.

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of source code.

LISTING 1: HEX DOWNLOAD CODE WRITTEN FOR MPLAB™-C17

```
void EraseFlash(void)
{
    rom int *EFp;                          // FLASH requires following sequence to
    unsigned int dataEF;                   // initiate a write

    EFp = (rom int *)0xd555;               // Setup pointer to D555h
    *EFp = 0xaaaa;                         // Write data AAAAh
    EFp = (rom int *)0xaaaa;
    *EFp = 0x5555;
    EFp = (rom int *)0xd555;
    *EFp = 0x8080;
    EFp = (rom int *)0xd555;
    *EFp = 0xaaaa;
    EFp = (rom int *)0xaaaa;
    *EFp = 0x5555;
    EFp = (rom int *)0xd555;
    *EFp = 0x1010;
    EFp = (rom int *)0x8000;
    do                                      // Wait for FLASH to erase
    {
        dataEF = *EFp;
        if(dataEF & 0x0020)
       Nop();
        Nop();
    } while(!(dataEF&0x0080));
    return;
}
void ProgPreamble(void)
{                                           // FLASH requires a preamble before each
    rom int *PPp;                           // word that is programmed

    PPp = (rom int *)0xd555;                // Setup pointer to D555h
    *PPp = 0xaaaa;                          // Write data AAAAh
    PPp = (rom int *)0xaaaa;
    *PPp = 0x5555;
    PPp = (rom int *)0xd555;
    *PPp = 0xa0a0;
    return;
}
char DownloadHex(void)
{
    unsigned char ByteCount,RecType,Checksum,FChecksum;
    unsigned char DHi,Errors;
    unsigned char bytes;
    unsigned int AddrL,AddrH;
    unsigned int HexData;
    unsigned char temp;
    char str[5];
    rom int *DHp;

    EraseFlash();                           // Erase FLASH
    AddrH = 0;                              // Make high address word 0
    while(1)                                //
    {                                       // Wait for a :
        while(1)
        {
            while(!DataRdyU2());
            if(RCREG2 == ':')
                break;
        }
        Errors = 0;                         // Preset errors to 0
        ByteCount = Hex8in();               // Read in ByteCount and store in Checksum
```

```
        Checksum = ByteCount;
        AddrL = Hex16in();                  // Read in low word of address and add
        Checksum += (unsigned char)AddrL;   // to Checksum
        Checksum += ((unsigned char)(AddrL>>8));
        RecType = Hex8in();                                 // Read in RecordType and add to Checksum
        Checksum += RecType;
        if(RecType == 0x00)                                 // Data record
        {
            if(AddrH)                                       // Assemble 16-bit word address
                DHp = (rom int *)((AddrL>>1)+0x8000);       // from AddrH and AddrL
            else
                DHp = (rom int *)(AddrL>>1);
            bytes = ByteCount>>1;                           // get number of words in record
            for(DHi=0;DHi<bytes;DHi++)                      // loop for number of words
            {
                 temp = Hex8in();                           // Read in word of data and
                HexData = (unsigned int)Hex8in();           // add to Checksum
                Checksum += temp;
                Checksum += (unsigned char)HexData;
                 HexData <<= 8;
                 HexData |= (unsigned int)temp;
                 if(DHp > (rom int *)0x3fff)                // If address in not in OTP
                 {                                          // then program
                    while(1)
                    {
                    ProgPreamble();                         // Program preamble
                    *DHp = HexData;                         // write cycle
                         while((HexData&0x0080) != (*DHp&0x0080))  // Wait for program cycle
                         Nop();                             // to terminate
                       if(*DHp == HexData)                  // Make sure data was programmed
                       break;                               // If not try to reprogram
                     }
                }
                DHp++;                                      // Increment address pointer
            }
            FChecksum = Hex8in();                           // Read in LineChecksum
            if(FChecksum != (~Checksum + 1))                // Compare to calculated
                Errors = 1;                                 // If not equal, increment errors
        }
        else if(RecType == 0x04)                            // Extended address record
        {
            AddrH = Hex16in();                              // Read in 16-bits of address
            Checksum += (unsigned char)AddrH;               // and add to Checksum
            Checksum += ((unsigned char)(AddrH>>8));
            FChecksum = Hex8in();                           // Read in Line Checksum
            if(FChecksum != (~Checksum + 1))                // Compare to calculated
                Errors = 1;                                 // If not equal, increment errors
        }
        else if(RecType == 0x01)                            // End of file record
        {
            FChecksum = Hex8in();                           // Read in LineChecksum
            if(FChecksum != (~Checksum + 1))                // Compare to calculated
                Errors = 1;                                 // If not equal, increment errors
            break;
        }
    }
    return Errors;                                          // Return number of errors
}
```

FIGURE 3: FLOWCHART

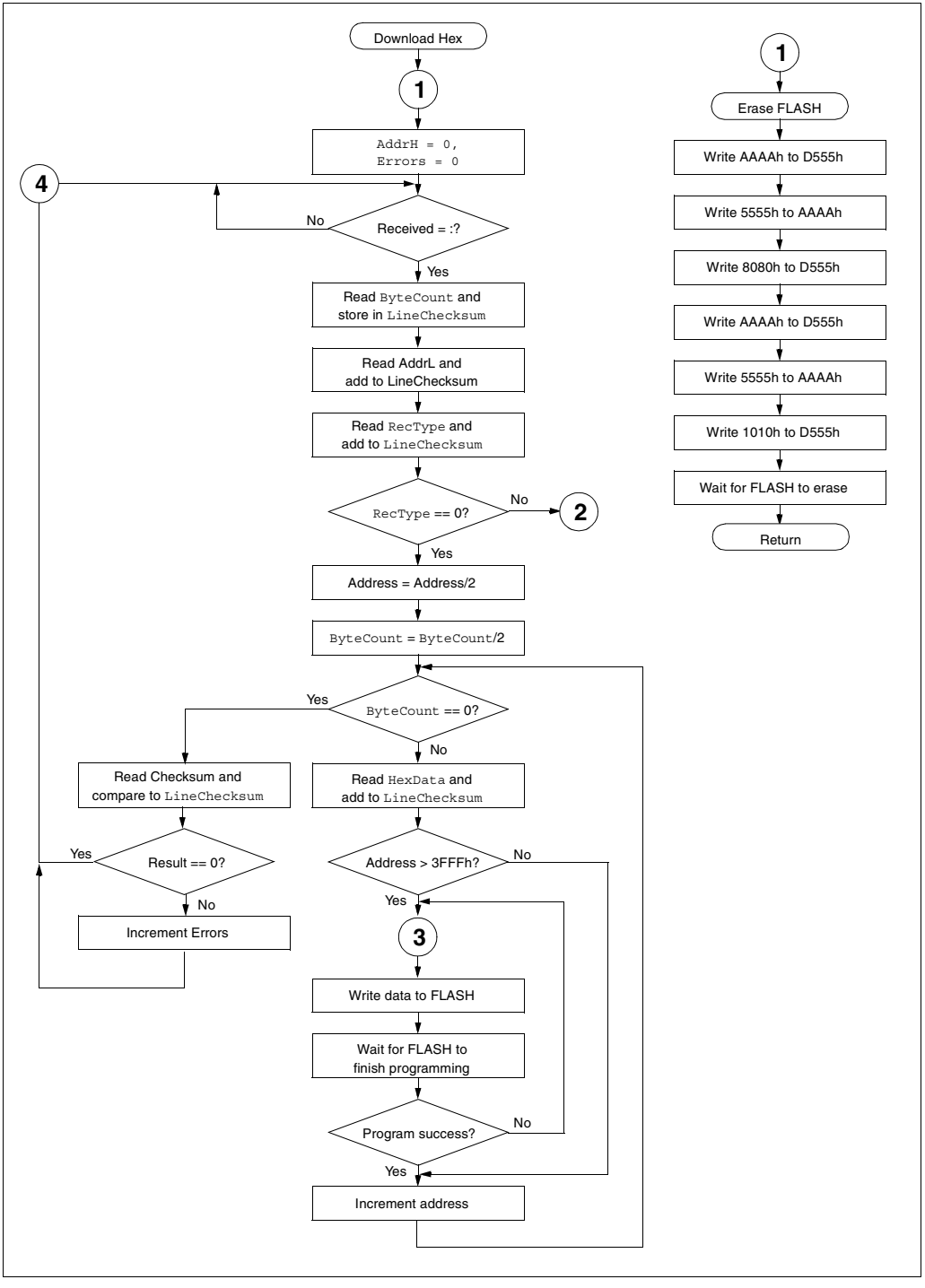
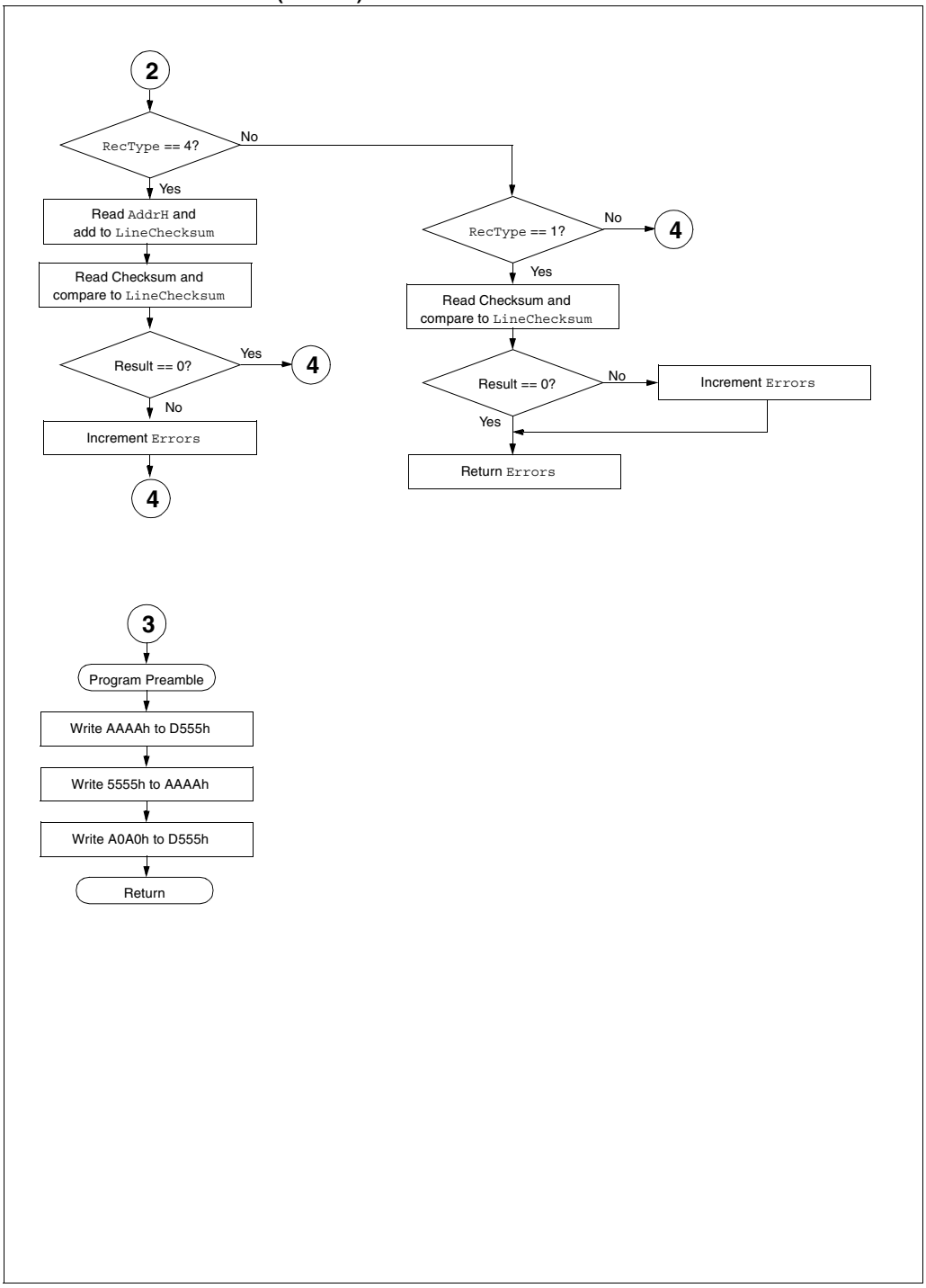


FIGURE 3 : FLOWCHART (CONT'D)



NOTES:

Downloading HEX Files to PIC16F87X PICmicro® Microcontrollers

Author: *Rodger Richey*
Microchip Technology Inc.

INTRODUCTION

The release of the PIC16F87X devices introduces the first mid-range family of devices from Microchip Technology that has the capability to read and write to internal program memory. This family has FLASH-based program memory, SRAM data memory and EEPROM data memory. The FLASH program memory allows for a truly reprogrammable system. Table 1 shows the features of the PIC16F87X family of devices.

ACCESSING MEMORY

The read and write operations are controlled by a set of Special Function Registers (SFRs). There are six SFRs required to access the FLASH program memory:

- EECON1
- EECON2
- EEDATA
- EEDATH
- EEADR
- EEADRH

The registers EEADRH:EEADR holds the 12-bit address required to access a location in the 8K words of program memory. The registers EEDATH:EEDATA are used to hold the data values. When reading program memory, the EEPGD bit (EECON1<7>) must be set to indicate to the microcontroller that the operation is going to be on program memory. If the bit is cleared, the operation will be performed on data memory at the address pointed to by EEADR. The EEDATA register will hold the data. The EECON1 register also has bits for write enable and to initiate the read or write operation. There is also a bit to indicate a write error has occurred, possibly due to a reset condition happening while a write operation is in progress. Figure 1 shows the register map for EECON1.

The EECON2 register is not a physical register. Reading it will result in all '0's. This register is used exclusively in the EEPROM and FLASH write sequences. Listing 1 shows the code snippet to initiate a write operation on the PIC16F87X devices.

TABLE 1: PIC16F87X FAMILY FEATURES

Key Features	PIC16F873	PIC16F874	PIC16F876	PIC16F877
Operating Frequency	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz
Resets	POR, BOR	POR, BOR	POR, BOR	POR, BOR
Flash Prog Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory	128	128	256	256
Interrupts	13	14	13	14
I/O Ports	Ports A,B,C	Ports A,B,C,D,E	Ports A,B,C	Ports A,B,C,D,E
Timers	3	3	3	3
Capture/Compare/PWM modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels

FIGURE 1: EECON1 REGISTER

R/W-x	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	—	—	—	WRERR	WREN	WR	RD
bit7							bit0

R= Readable bit
W= Writable bit
S= Settable bit
U= Unimplemented bit, read as '0'
- n= Value at POR reset

bit 7: **EEPGD**: Program / Data EEPROM Select bit
1 = Accesses Program memory
0 = Accesses data memory
Note: This bit cannot be changed while a write operation is in progress.

bit 6:4: **Unimplemented:** Read as '0'

bit 3: **WRERR**: EEPROM Error Flag bit
1 = A write operation is prematurely terminated
(any $\overline{\text{MCLR}}$ reset or any WDT reset during normal operation)
0 = The write operation completed

bit 2: **WREN**: EEPROM Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1: **WR**: Write Control bit
1 = initiates a write cycle.
The bit is cleared by hardware once write is complete.
The WR bit can only be set (not cleared) in software.
0 = Write cycle to the EEPROM is complete

bit 0: **RD**: Read Control bit
1 = Initiates an EEPROM read (read takes one cycle)
RD is cleared in hardware. The RD bit can only be set (not cleared) in software.
0 = Does not initiate an EEPROM read

HEX FILE FORMAT

The data to be programmed into program memory will be read into the microcontroller using one of its standard interface modules: SPI, I²C™, USART, or PSP. Probably the simplest format to send the data to the microcontroller is in the standard HEX format used by the Microchip development tools. The formats supported are the Intel HEX Format (INHX8M), Intel Split HEX Format (INHX8S), and the Intel HEX 32 Format (INHX32). The most commonly used formats are the INHX8M and INHX32 and therefore are the only formats discussed in this document. Please refer to Appendix A in the MPASM User's Guide (DS33014) for more information about HEX file formats. The difference between INHX8M and INHX32 is that INHX32 supports 32-bit addresses using a linear address record. The basic format of the hex file is the same between both formats as shown below:

```
:BBAAAATTHHHH...HHHHCC
```

Each data record begins with a 9 character prefix and always ends with a 2 character checksum. All records begin with a '**:**' regardless of the format. The individual elements are described below.

- **BB** - is a two digit hexadecimal byte count representing the number of data bytes that will appear on the line. Divide this number by two to get the number of words per line.
- **AAAA** - is a four digit hexadecimal address representing the starting address of the data record. Format is high byte first followed by low byte. The address is doubled because this format only supports 8-bits (to find the real PICmicro address, simply divide the value **AAAA** by 2).
- **TT** - is a two digit record type that will be '00' for data records, '01' for end of file records and '04' for extended address record (INHX32 only).
- **HHHH** - is a four digit hexadecimal data word. Format is low byte followed by high byte. There will be **BB**/2 data words following **TT.**
- CC - is a two digit hexadecimal checksum that is the two's complement of the sum of all the preceding bytes in the line record.

Since the PIC16F87X devices only have a maximum of 8K words, the linear address record '04' is ignored by the routine. The HEX file is composed of ASCII characters 0 thorough 9 and A to F and the end of each line has a carriage return and linefeed. The downloader code in the PICmicro microcontrollers must convert the ASCII characters to binary numbers to be used for programming.

PICmicro Code

The sample downloader code does not specifically use one of the interface modules on the PIC16F87X device. Instead, a routine called GetByte retrieves a single character from the HEX file over the desired interface. It is up to the engineer to write this routine around the desired interface. Another routine GetHEX8 calls GetByte twice to form a two digit hexadecimal number.

One issue that arises is how many times to reprogram a location that does not program correctly. The sample code provided simply exits the downloader routine and stores a value of 0xFF in the WREG if a program memory location does not properly program on the first attempt. The engineer may optionally add code to loop several times if this event occurs.

Still another issue that is not specifically addressed in the sample code is to prevent the downloader from overwriting its own program memory address locations. The designer must add an address check to prevent this situation from happening.

Finally, the designer must account for situations where the download of new code into the microcontroller is interrupted by an external event such as power failure or reset. The system must be able to recover from such an event. This is not a trivial task, is very system dependent, and is therefore left up to the designer to provide the safeguards and recovery mechanisms.

Another error that could happen is a line checksum error. If the calculated line checksum does not match the line checksum from the HEX file, a value of 1 is returned in WREG. The part of the routine that calls the downloader should check for the errors 0xFF (could not program a memory location) and 1. If program memory is programmed correctly and no errors have been encountered, the downloader routine returns a 0 in WREG to indicate success to the calling routine. Figure 2 shows the flowchart for the downloader routines. Listing 2 shows the complete listing for the downloader code.

The routine ASCII2HEX converts the input character to a binary number. The routine does not provide any out of range error checking for incoming characters. Since the only valid characters in a HEX file are the colon (:), the numbers 0 through 9 and the letters A through F, the routine can be highly optimized. It first subtracts 48 from the character value. For the ASCII numbers 0 through 9, this results in a value from 0 to 9. If the character is A through F, the result is a number greater than 15. The routine checks to see if the upper nibble of the result is 0. If not 0, then the original value was A through F and the routine now subtracts an additional 43 from the character resulting in the binary values 10 through 15. The colon is not accounted for in this routine because the main part of the downloader code uses it as a line sync.

LISTING 1: FLASH WRITE SEQUENCE

```
bsfSTATUS,RP1      ; Bank2
bcfSTATUS,RP0
movfAddrH,W         ; Load address into
movwfEEADRH          ; EEADRH:EEADR
movfAddrL,W
movwfEEADR
bsfSTATUS,RP0      ; Bank3
bsfEECON1,EEPGD    ; Set for Prog Mem
bsfEECON1,RD       ; read operation
bcfSTATUS,RP0      ; Bank2
nop
movfEEDATA,W        ; Data is read
…                       ; user can now
movfEEDATH,W        ; access memory
…
```

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of source code.

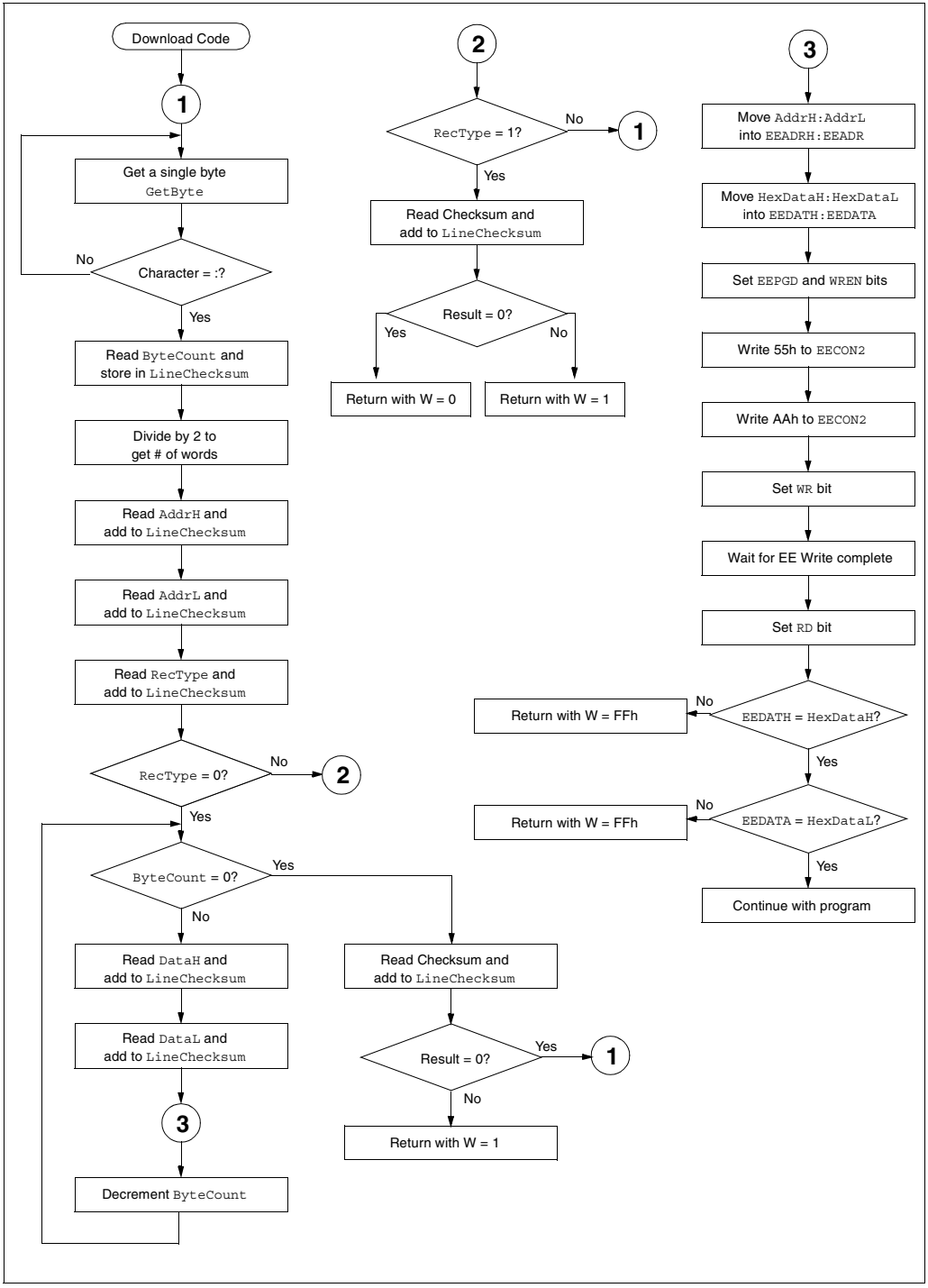
LISTING 2: HEX DOWNLOAD CODE WRITTEN FOR MPASM

```
        list p=16f877
        #include "c:\progra~1\mplab\p16f877.inc"


DownloadCode                            ;Uses USART to receive data from PC
        banksel     RCREG
DCStart
        call        GetByte
        movlw       ':'                 ;Wait for colon
        subwf       RCREG,W
        btfss       STATUS,Z
        goto        DCStart

        call        GetHex8             ;Read byte count
        movwf       ByteCount           ;Store in register
        movwf       LineChecksum        ;Store in line checksum
        bcf         STATUS,C
        rrf         ByteCount,F         ;Divide byte counter by 2 to get words

        call        GetHex8             ;Read high byte of 16-bit address
        movwf       AddrH
        addwf       LineChecksum,F      ;Add high byte to line checksum
        call        GetHex8             ;Read low byte of 16-bit address
        movwf       AddrL
        addwf       LineChecksum,F      ;Add low byte to line checksum

        call        GetHex8             ;Read record type
        movwf       RecType
        addwf       LineChecksum,F      ;Add to line checksum

DataRec                                 ;Data reception
        movf        RecType,F           ;Check for data record (0h)
        btfss       STATUS,Z
        goto        EndOfFileRec        ;Otherwise check for EOF
DRLoop
        movf        ByteCount,F         ;Check for bytecount = 0
        btfsc       STATUS,Z
        goto        DRCkChecksum        ;If zero, goto checksum validation
        call        GetHex8             ;Read lower byte of data (2 characters)
        movwf       HexDataL            ;Add received data to checksum
        addwf       LineChecksum,F
        call        GetHex8             ;Read upper byte of data (2 characters)
        movwf       HexDataH            ;Add received data to checksum
        addwf       LineChecksum,F

WriteDataSequence                       ;Write sequence to internal prog. mem FLASH
        banksel     EEADRH
        movf        AddrH,W             ;Write address to EEADRH:EEADR registers
        movwf       EEADRH
        movf        AddrL,W
        movwf       EEADR
        movf        HexDataH,W          ;Write data to EEDATH:EEDATA registers
        movwf       EEDATH
        movf        HexDataL,W
        movwf       EEDATA
        banksel     EECON1              ;Write sequence
        bsf         EECON1,EEPGD        ;Set EEPGD to indicate program memory
        bsf         EECON1,WREN         ;Enable writes to memory
        bcf         INTCON,GIE          ;Make sure interrupts are disabled
        movlw       0x55                ;Required write sequence
        movwf       EECON2
        movlw       0xaa
```

```
        movwf       EECON2
        bsf         EECON1,WR           ;Start internal write cycle
        nop
        nop
        bcf         EECON1,WREN         ;Disable writes

        banksel     EECON1              ;Read sequence
        bsf         EECON1,EEPGD        ;Set EEPGD to indicate program memory
        bsf         EECON1,RD           ;Enable reads from memory
        bcf         STATUS,RP0
        nop
        movf        EEDATH,W            ;Compare memory value to HexDataH:HexDataL
        subwf       HexDataH,W
        btfss       STATUS,Z
        retlw       0xff                ;If upper byte not equal, return FFh
        movf        EEDATA,W            ; to indicate programming failure
        subwf       HexDataL,W
        btfss       STATUS,Z
        retlw       0xff                ;If lower byte not equal, return FFh
                                        ; to indicate programming failure
        incf        AddrL,F             ;Increment address for next iteration
        btfsc       STATUS,Z
        incf        AddrH,F
        decf        ByteCount,F         ;Decrement byte count
        goto        DRLoop              ;Go back to check for ByteCount = 0

DRCkChecksum                            ;Checksum verification
        call        GetHex8             ;Read in checksum
        addwf       LineChecksum,W      ;Add to calculated checksum
        btfss       STATUS,Z            ;Result should be 0
        retlw       1                   ; If not return 1 to indicate checksum fail
        goto        DCStart             ;Do it again

EndOfFileRec                           ;End of File record (01h)
        decf        RecType,W          ;If EOF record, decrement should = 0
        btfss       STATUS,Z
        goto        DCStart            ;Not valid record type, wait for next :
        call        GetHex8            ;Read in checksum
        addwf       LineChecksum,W     ;Add to calculated checksum
        btfss       STATUS,Z           ;Result should be 0
        retlw       1                  ; If not return 1 to indicate checksum fail
        retlw       0                  ;Otherwise return 0 to indicate success


GetByte
; Insert your code here to retrieve a byte of data from
; the desired interface. In this case it is the USART on F877.
;clear CTS
;       banksel    PIR1
;GH4Waitbtfss      PIR1,RCIF
;       goto       GH4Wait
;set CTS
        nop
        banksel     RCREG
        movf        RCREG,W
        return

GetHex8                                 ;This function uses the USART
        call        GetByte             ;Read a character from the USART
        call        ASCII2Hex           ;Convert the character to binary
        movwf       Temp                ;Store result in high nibble
        swapf       Temp,F

        call        GetByte             ;Read a character from the USART
        call        ASCII2Hex           ;Convert the character to binary
        iorwf       Temp,F              ;Store result in low nibble
```

```
        movf        Temp,W          ;Move result into WREG
        return

ASCII2Hex                           ;Convert value to binary
        movwf       Temp1           ;Subtract ASCII 0 from number
        movlw       '0'
        subwf       Temp1,F
        movlw       0xf0            ;If number is 0-9 result, upper nibble
        andwf       Temp1,W         ; should be zero
        btfsc       STATUS,Z
        goto        ASCIIOut
        movlw       'A'-'0'-0x0a    ;Otherwise, number is A - F, so
        subwf       Temp1,F         ;subtract off additional amount
ASCIIOut
        movf        Temp1,W         ;Value should be 0 - 15
        return

        end
```

FIGURE 2: FLOWCHART



NOTES:

Calculating Program Memory Checksums Using a PIC16F87X

Author: *Rodger Richey*
Microchip Technology Inc.

INTRODUCTION

Many applications require the microcontroller to calculate a checksum on the program memory to determine if the contents have been corrupted. Until now, the only family of PICmicro® microcontrollers to have the capability to read from program memory are the PIC17CXXX devices. The PIC16F87X devices are the first 14-bit core PICmicro microcontrollers that are able to access program memory in the same fashion as used with data EEPROM memory. These devices are FLASH extensions of the popular PIC16C7X family. Table 1 shows a comparison between the two PICmicro microcontroller families.

TABLE 1: PIC16C7X vs. PIC16F87X

Feature	PIC16C7X	PIC16C87X
Pins	28 or 40	28 or 40
Timers	3	3
Interrupts	11 or 12	13 or 14
Communication	PSP, USART, SSP (SPI or I^2C Slave)	PSP, USART, SSP(SPI or I^2C Master/Slave)
Frequency	20 MHz	20 MHz
A/D	8-bit	10-bit
CCP	2	2
Program Mem.	4K or 8K EPROM	4K or 8K FLASH
RAM	192 or 368 bytes	192 or 368 bytes
Data EEPROM	None	128 or 256 bytes
Other	---	In-Circuit Debugger

ACCESSING MEMORY

The data EEPROM and FLASH Program memory are both accessed using the same method. An address and/or data value are stored in Special Function Registers (SFR) and then memory is accessed using control bits in other SFRs. There are six SFRs required to access memory:

- EECON1
- EECON2
- EEDATA
- EEDATH
- EEADR
- EEADRH

When interfacing to data EEPROM memory, the address is stored in the EEADR register and the data is accessed using the EEDATA register. The operation is controlled using the EECON1 and EECON2 registers. The register map for EECON1 is shown in Figure 1. EECON2 is not a physical register. Reading it will result in all '0's. This register is used exclusively in the EEPROM and FLASH write sequences.

When interfacing to FLASH program memory, the address is stored in the EEADRH:EEADR registers and the data is accessed using the EEDATH:EEDATA registers. Since the same set of control registers are used to access data and program memory, the EEPGD bit (EECON1<7>) is used to indicate to the microcontroller whether the operation is going to be on data memory (EEPGD = 0) or program memory (EEPGD = 1). Refer to Section 7.0 in the PIC16F87X data sheet (DS30292) for more information about using the EEPROM and FLASH memories.

FIGURE 1: EECON1 REGISTER

R/W-x	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	—	—	—	WRERR	WREN	WR	RD
bit7							bit0

R = Readable bit
W = Writable bit
S = Settable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **EEPGD**: Program / Data EEPROM Select bit
1 = Accesses Program memory
0 = Accesses data memory
Note: This bit cannot be changed while a write operation is in progress.

bit 6:4: **Unimplemented:** Read as '0'

bit 3: **WRERR**: EEPROM Error Flag bit
1 = A write operation is prematurely terminated
(any $\overline{\text{MCLR}}$ reset or any WDT reset during normal operation)
0 = The write operation completed

bit 2: **WREN**: EEPROM Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1: **WR**: Write Control bit
1 = initiates a write cycle. (The bit is cleared by hardware once write is complete.
The WR bit can only be set (not cleared) in software.
0 = Write cycle to the EEPROM is complete

bit 0: **RD**: Read Control bit
1 = Initiates an EEPROM read (read takes one cycle. RD is cleared in hardware.
The RD bit can only be set (not cleared) in software).
0 = Does not initiate an EEPROM read

HEX FILE FORMAT

Development tools from Microchip support the Intel HEX Format (INHX8M), Intel Split HEX Format (INHX8S), and the Intel HEX 32 Format (INHX32). The most commonly used formats are the INHX8M and the INHX32. These are the only formats discussed in this document. Please refer to Appendix A in the MPASM User's Guide (DS33014) for more information about HEX file formats. The difference between INHX8M and INHX32 is that INHX32 supports 32-bit addresses using a linear address record. The basic format of the hex file is the same between INHX8M and INHX32 as shown below:

:BBAAAATTHHHH...HHHHCC

Each data record begins with a 9 character prefix and always ends with a 2 character checksum. All records begin with a ':' regardless of the format. The individual elements are described below.

- **BB** - is a two digit hexadecimal byte count representing the number of data bytes that will appear on the line.
- **AAAA** - is a four digit hexadecimal address representing the starting address of the data record. Format is high byte first followed by low byte, the address is doubled because this format only supports 8-bits (to find the real PICmicro address, simply divide the value **AAAA** by 2).
- **TT** - is a two digit record type that will be '00' for data records, '01' for end of file records and '04' for extended address record (INHX32 only).
- **HHHH** - is a four digit hexadecimal data word. Format is low byte followed by high byte. There will be **BB**/2 data words following **TT.**
- **CC** - is a two digit hexadecimal checksum that is the two's complement of the sum of all the preceding bytes in the line record.

HEX File Preparation

The checksum used to verify program memory contents is a 14-bit number calculated only on the program memory contents of a HEX file. The reason that only 14-bits is used is because the PIC16F87X has 14-bit wide program memory.

The first step to obtaining the checksum is to get a complete HEX file that has all address locations specified. This can be easily accomplished in MPLAB by enabling the programmer, either PROMATE II or PICSTART PLUS, whichever one is available. Load the HEX file into MPLAB using the menus File -> Import -> Download to Memory. Then save the HEX file using File -> Export -> Save HEX File. Make sure that the Program Memory box is checked with a range of 0 to 8191 and the Configuration bits and IDs box are also checked. It is optional to check the EEPROM memory box depending on you application. This will create a complete HEX file including all program memory, configuration word, IDs, and optionally EEPROM memory.

The checksum provided by a programmer, such as PROMATE II or PICSTART PLUS, is not valid because the configuration word and device ID are included in the calculation. Therefore, a different program is required to calculate the program memory checksum. Once a complete HEX file has been obtained by the previously presented method, it must be processed and modified to contain the checksum. The program CHECKSUM.EXE, which is a DOS based program, reads in the HEX file, calculates the checksum, and outputs the new HEX file with checksum included. The checksum is calculated by:

1. Adding together the memory locations 0x0000 to 0x1FFE.
2. Mask off all but the lower 14-bits.
3. Take the 2's complement of Step 2.
4. Mask off all but the lower 14-bits.
5. Save this value into the HEX file at address 0x1FFF.

The program ignores all configuration word, ID, and EEPROM memory information in the HEX file and dumps it to the output file unchanged. The output file can then be programmed into the PIC16F87X device.

PICmicro Code

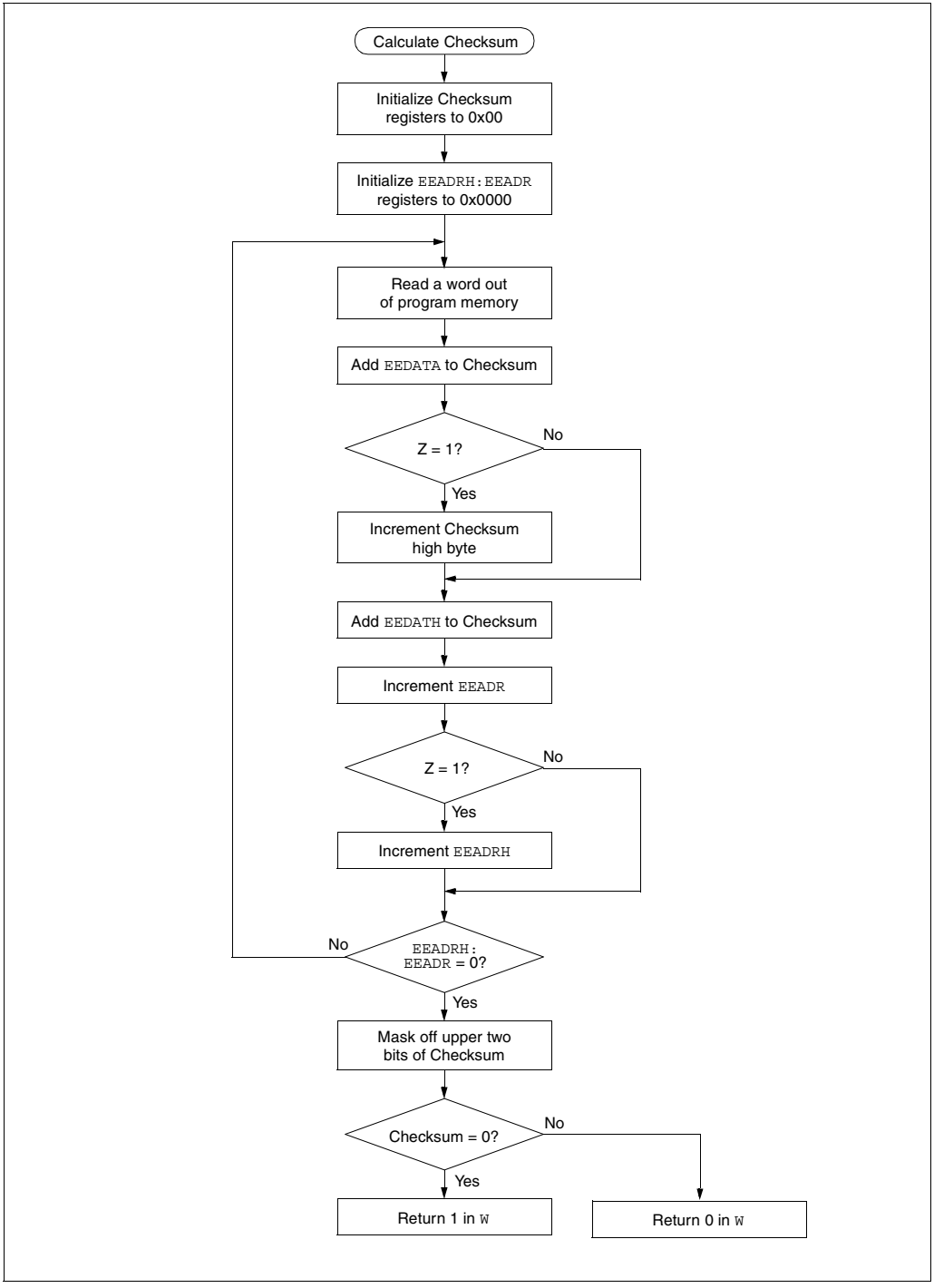
The code used by the PIC16F87X to calculate checksum uses 36 words of program memory and two data memory locations. The example code uses data memory locations 0x7E and 0x7F to store the calculated checksum. These locations are shared across all banks. The user can optionally change these locations and add banking into the routine. Figure 2 shows the flowchart for the routine. The checksum is created such that by adding up all program memory locations, a 14-bit result of 0x0000 is obtained. Since the calculation is done in 16-bits, the result will actually be 0x4000, but the upper two bits are masked off by the routine. Example 1 shows the code in MPASM to calculate the program memory checksum. If the program memory verifies, the routine returns a '1'. If a failure is detected, the routine returns a '0'.

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of source code.

LISTING 1: PROGRAM MEMORY CHECKSUM ROUTINE

```
CalcChecksum
    bsf     STATUS,RP1      ;Go to Bank 2
    bcf     STATUS,RP0
    clrf    ChecksumL       ;Clear the Checksum
    clrf    ChecksumH       ;registers
    clrf    EEADR           ;Set the Program Memory
    clrf    EEADRH          ; address to 0x0000
CLoop                       ;Loop for each location
    bsf     STATUS,RP0      ; to read memory location
    bsf     EECON1,EEPGD    ;Set for program memory
    bsf     EECON1,RD       ;Set for read operation
    bcf     STATUS,RP0      ;Go to Bank 2
    nop
    movf    EEDATA,W        ;Add low byte to Checksum
    addwf   ChecksumL,F
    btfsc   STATUS,C        ;Check for overflow
    incf    ChecksumH,F     ;Yes, increment Checksum
    movf    EEDATH,W        ;Add high byte
    addwf   ChecksumH,F
    incf    EEADR,F         ;Increment low address
    btfsc   STATUS,Z        ;Check for overflow
    incf    EEADRH,F        ;Increment high address
    movf    EEADRH,F        ;Check to see if
    btfss   STATUS,Z        ; address wrapped
    goto    CLoop           ; from 0x1fff to
    movf    EEADR,F         ; 0x0000
    btfss   STATUS,Z
    goto    CLoop
                            ;Checkcum calculation complete
    bcf     ChecksumH,7     ;Clear upper 2 bits
    bcf     ChecksumH,6     ; only 14-bit checksum
    movf    ChecksumH,F     ;Checksum should be 0
    btfss   STATUS,Z
    retlw0                  ;Checksum failed
    movf    ChecksumL,F
    btfss   STATUS,Z
    retlw0                  ;Checksum failed
    retlw1                  ;Checksum passed
```

FIGURE 1: FLOWCHART



NOTES:

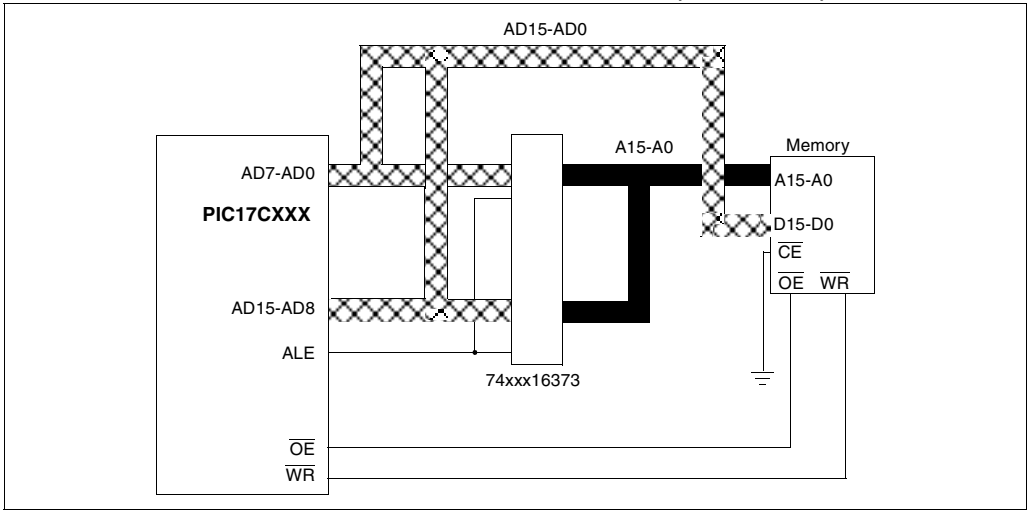
Simplifying External Memory Connections of PIC17CXXX PICmicro® Microcontrollers

Author: *Rodger Richey*
Microchip Technology Inc.

INTRODUCTION

The PIC17CXXX family of PICmicro® microcontrollers has an external program memory interface. Since the PIC17CXXX devices implement a 16-bit instruction word, the external memory must be 16-bits wide. The addressing space of these devices is 64K x 16, which requires 16-bits of address as well. Until a few years ago, the designer had to use two 8-bit latches for addressing and two 8-bit wide memories. Now, many manufacturers of logic and memory devices have developed 16-bit wide devices. These new 16-bit wide devices can simplify the layout, reduce part count and cost as shown in Figure 1.

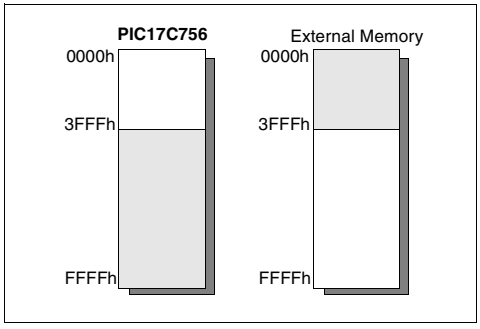
FIGURE 1: EXTERNAL MEMORY INTERFACE BLOCK DIAGRAM (x16 DEVICES)



IMPLEMENTATION

Although EPROM and static RAM devices are compatible with PICmicro microcontrollers, FLASH memory was chosen to implement a reprogrammable system. Due to the operational characteristics of FLASH memory, the PIC17CXXX device must be configured in extended microcontroller mode to implement an external reprogrammable system. In this mode, the internal memory of the PICmicro microcontroller is operational, and the remainder of the 64K memory is external to the device (see Figure 2). The bootloader routine is located in the on-chip memory. This routine reads data from the outside world and programs it into the FLASH memory. The PIC17CXXX has many interfaces, which could be used for downloading new code into the external memory including: USART, SPI, or I^2C™. Since the PIC17CXXX has two USARTs, one could be used to communicate with other devices in the system and the second USART could be used for downloading new code into the FLASH.

FIGURE 2: PIC17C756 IN EXTENDED MICROCONTROLLER MODE



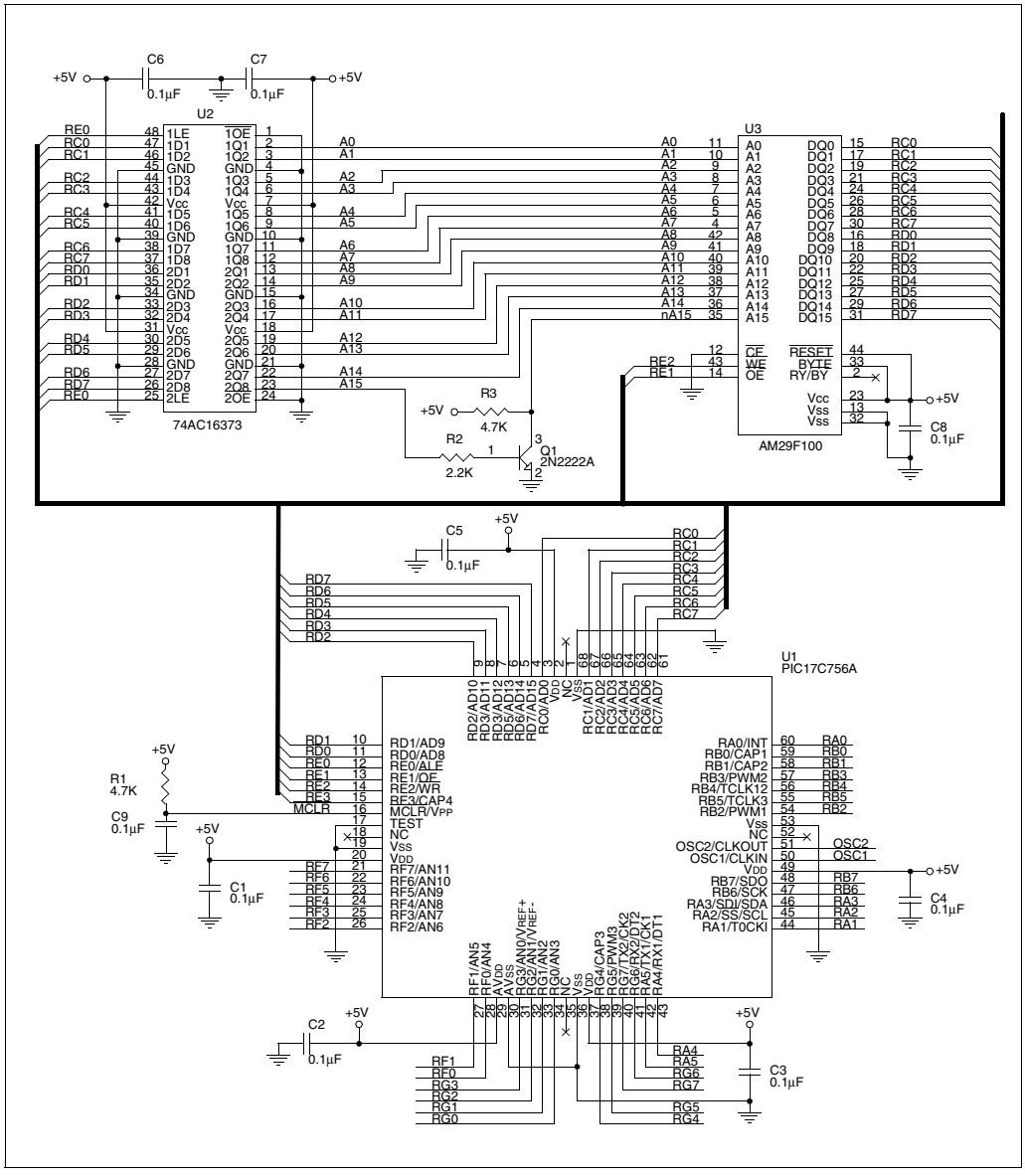
LOGIC DEVICES

In recent years, many manufacturers have developed x16 versions of the more popular 74xxx devices. These new devices use the following naming convention:

74xxx16yyy

where xxx defines the technology (HC,AC,FCT, etc.) and yyy defines the part number (244, 373, etc.). For the purposes of this design, the 74xxx16373 will be used. The technology depends on the operating frequency of the microcontroller. Appendix A lists some of the manufacturers of the x16 logic devices. All these devices have the same pinout which includes 16 inputs, 16 outputs, two latch enable (LE) pins and two output enable (OE) pins. Each set of LE/OE controls 8-bits of input/output. As shown in Figure 3, both LE pins are tied to the RE0/ALE pin of the microcontroller and the OE pins are tied to ground. These devices can be found in various packages types (DIP through TSSOP).

FIGURE 3: SCHEMATIC USING x16 DEVICES



MEMORY DEVICES

Almost all manufacturers of memory make a x16 device. Currently, the smallest x16 FLASH memories are the 1 Megabit (64K x 16) devices from AMD or Catalyst. Memory selection should address the required program voltage requirements because some manufacturers have single supply devices and others have multiple supply devices. Appendix A lists some of the manufacturers of x16 FLASH memory devices. Some of the basic features of any FLASH memory are:

- Power supply options
  - Single power supply for read, erase and program operations (desirable for 5V only systems)
  - or Dual power supply, one for read an another for program/erase operations
- Access time
- Software method of detecting end of program cycle
- Full chip erase capability
- Hardware and software data protection

The devices from AMD provide many superior features over other manufacturers of FLASH memory including:

- x8 or x16 configurable
- Low power consumption:
  - 28 mA typical active read current
  - 30 mA typical program/erase current
  - 25 μA typical standby current
- Any combination of sectors can be erased
- Embedded algorithms that automatically pre-programs and erases sectors or programs and verifies data at a specified address
- Minimum 100,000 program/erase cycles
- JEDEC compatible pinout and software commands
- Hardware pin for detecting end of program/erase cycles
- Software commands that suspend or resume an erase cycle to read data out of other sectors
- Hardware reset pin

Figure 3 shows the schematic of the PIC17C756A with the AM29F100 FLASH memory from AMD. The first thing to notice is that the address line, A15, is inverted before going to the FLASH. The reason is that commands must be issued to the lower half of the FLASH to program or erase. The commands for both program and erase are shown below.

Program

1. Send AAAAh to address 5555h
2. Send 5555h to address 2AAAh
3. Send 8080h to address 5555h
4. Send 16-bits of data to desired address

Chip Erase

1. Send AAAAh to address 5555h
2. Send 5555h to address 2AAAh
3. Send 8080h to address 5555h
4. Send AAAAh to address 5555h
5. Send 5555h to address 2AAAh
6. Send 1010h to address 5555h

Sector Erase

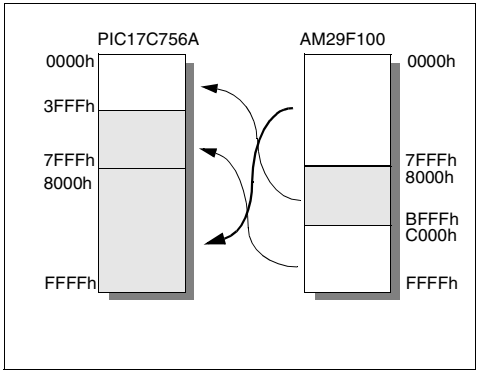
1. Send AAAAh to address 5555h
2. Send 5555h to address 2AAAh
3. Send 8080h to address 5555h
4. Send AAAAh to address 5555h
5. Send 5555h to address 2AAAh
6. Send 3030h to desired sector address

As shown in Figure 2, the on-chip memory of the PICmicro microcontroller is located in the first part of program memory, and in some devices, this on-chip program memory overlaps the address 2AAAh in the FLASH. The length of this memory depends on the device:

- PIC17C42A from 0000h to 07FFh,
- PIC17C43 from 0000h to 0FFFh,
- PIC17C44 from 0000h to 1FFFh,
- PIC17C752 from 0000h to 1FFFh,
- PIC17C756A from 0000h to 3FFFh,
- PIC17C762 from 0000h to 1FFFh, and
- PIC17C766 from 0000h to 3FFFh

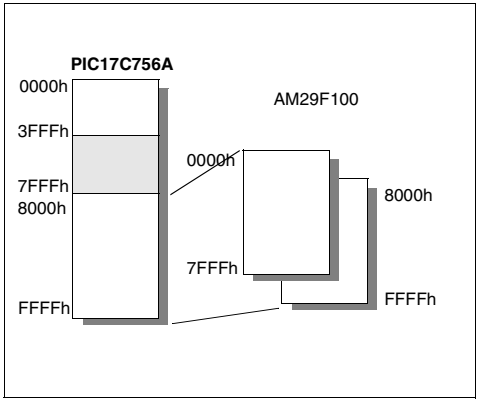
The first three devices do not require the inverter circuit, but the PIC17C756A does. When the bootloader program tries to send any data to address 2AAAh, the core thinks it is writing to the internal memory address because it is less than 3FFFh. The address/data lines from the PICmicro microcontroller assert the correct address/data, but the control lines do not allow the FLASH memory to be enabled for reading or writing. Therefore, by inverting A15, address 2AAAh of the FLASH can be accessed at location AAAAh in the program memory of the PIC17C756A. Address 5555h can be accessed at location D555h. The PIC17C756A is now able to send program and erase commands to the FLASH. The program memory address locations 4000h to 7FFFh are now located in the FLASH at locations C000h to FFFFh (See Figure 4).

FIGURE 4: OVERLAP OF FLASH



The previous method leaves part of the external FLASH unused, which can amount to 25% of program memory depending on the PICmicro microcontroller used. There is another method which completely uses all of the FLASH, but requires additional firmware support. The inverter circuit shown in Figure 3 is connected to the CE pin of the FLASH and A15 of the FLASH is connected to an I/O pin of the PICmicro microcontroller. In this case, external program memory becomes "banked". Only addresses from 8000h to FFFFh can be used and the I/O pin controls which bank is selected (See Figure 5). The designer must now have some previous knowledge about the locations of routines in external memory. A table must be created to define the location of each routine in external memory by bank (0 or 1) and address (8000h to FFFFh). This table is located in the on-chip program memory of the PICmicro microcontroller and cannot be changed. The designer must be careful when constructing this table to take into account the growth routines of old routines and addition of new routines for bug fixes or enhancements. In most cases, the C compiler cannot correctly execute code from both banks of FLASH. One bank would be for data and the other for code.

FIGURE 5: BANKING OF FLASH



CONCLUSION

By using the new x16 logic and memory devices, a designer can lower part count, cut cost, reduce board size and simplify layout. Since most manufacturers use the standard JEDEC footprint for their devices, single source supply concerns can be eliminated.

APPENDIX A

The following list of manufacturers is provided for reference only and is not meant to be a complete listing of all companies producing x16 logic and memory products.

x16 Logic Manufacturers

- National Semiconductor www.national.com
- IDT www.idt.com
- Quality Semiconductor www.qualitysemi.com
- Pericom Semiconductor www.pericom.com
- Texas Instruments www.ti.com

x16 FLASH Manufacturers

- AMD www.amd.com
- Catalyst www.catsemi.com
- Hyundai www.hea.com
- Intel www.intel.com
- ISSI www.issiusa.com
- Micron Technology www.micron.com/flash
- Texas Instruments www.ti.com

Technique to Calculate Day of Week

Author: *Tan Beng Hai*
Microchip Technology Inc.

INTRODUCTION

Basically, there are two kinds of electronic systems that come with a built-in calendar. The first kind of system is used mainly to display a calendar for a user's convenience. Examples of these systems are digital watches, computers, VCRs and TVs with on-screen display features. The second kind of system is required to know whether a given date is a weekend or weekday. Examples of such a system are multi-rate meters (such as a phone bill meter) and electronic pricing systems, where the weekend rate and the weekday rate are different.

In order to build an electronic calendar into the system, the designer needs to write a piece of software that will be able to determine the day (Sunday, Monday, ….Saturday) of the week when a date is input into the system. This routine is the basic component of an electronics calendar.

This Technical Brief provides a technique to find the exact day for a given date input.

THEORY OF CALCULATION

The method used to calculate the day of week is a straight forward and simple one. This method makes use of 31 December 1989 as a reference point. The reason why this date was chosen as a reference point is because it was on the last day of the week (Sunday), and also on the last day of a year (this makes it easy to calculate the number of days since the next day/date will be first/first). One day after this date was Monday, and two days after this date was Tuesday, and so on.

The date given will be N days after the 31st December 1989, and if the number N is divided by the number of days in a week (7), the return will correspond to a specific day of the week. For this application, 0 corresponds to Sunday, 1 to Monday, and so on till 6 corresponds to Saturday. Therefore, when a date is given, the number of days from the date given after the 31 December can be calculated. Then the division of the number by 7 will give a remainder, which will correspond to the day of the week that the system required.

DESCRIPTION OF SOFTWARE

This application note provides two routines for the calculation of the day of the week. One is written in ANSI C and the other is written in assembly language using a PIC16C54 microcontroller.

In these routines, the number of days after 31 December 1989 is calculated and stored in a register called `AccValue`. There are a total of three steps involved in getting the number of days.

The first step is to find out the difference in years, and convert the difference into days. A 16-bit counter, `TempYear`, is used as a temporary counter and is initialized to the year 1990. The routine will keep comparing the contents of `TempYear` to the year (`CurrentYear` in the 'C' software) that is input in the software. If the `TempYear` value is less than the year value, `TempYear` will be increased by 1 until the contents of `TempYear` match the year given.

The `AccValue` will be increased by 1 (instead of 365, because 365MOD7 = 1) or by 2 (if `TempYear` is a leap year) for each comparison in which the `TempYear` value is less than the year value.

The second step involves calculating the number of days that have elapsed between the first day (inclusive) of that year and the first day of the month given. The number of days elapsed is calculated and pre-stored in a table. This value is retrieved with respect to the input month and added to the `AccValue`.

The way to calculate the pre-stored value is as follows; for January there is less than one month elapsed, therefore the value stored is 0. For February, the month passed is only 1 (January) and the value stored is 31MOD7, which is 3. For March, 2 months have passed, (January and February), so the value stored is (28+31)MOD7, which is 3.

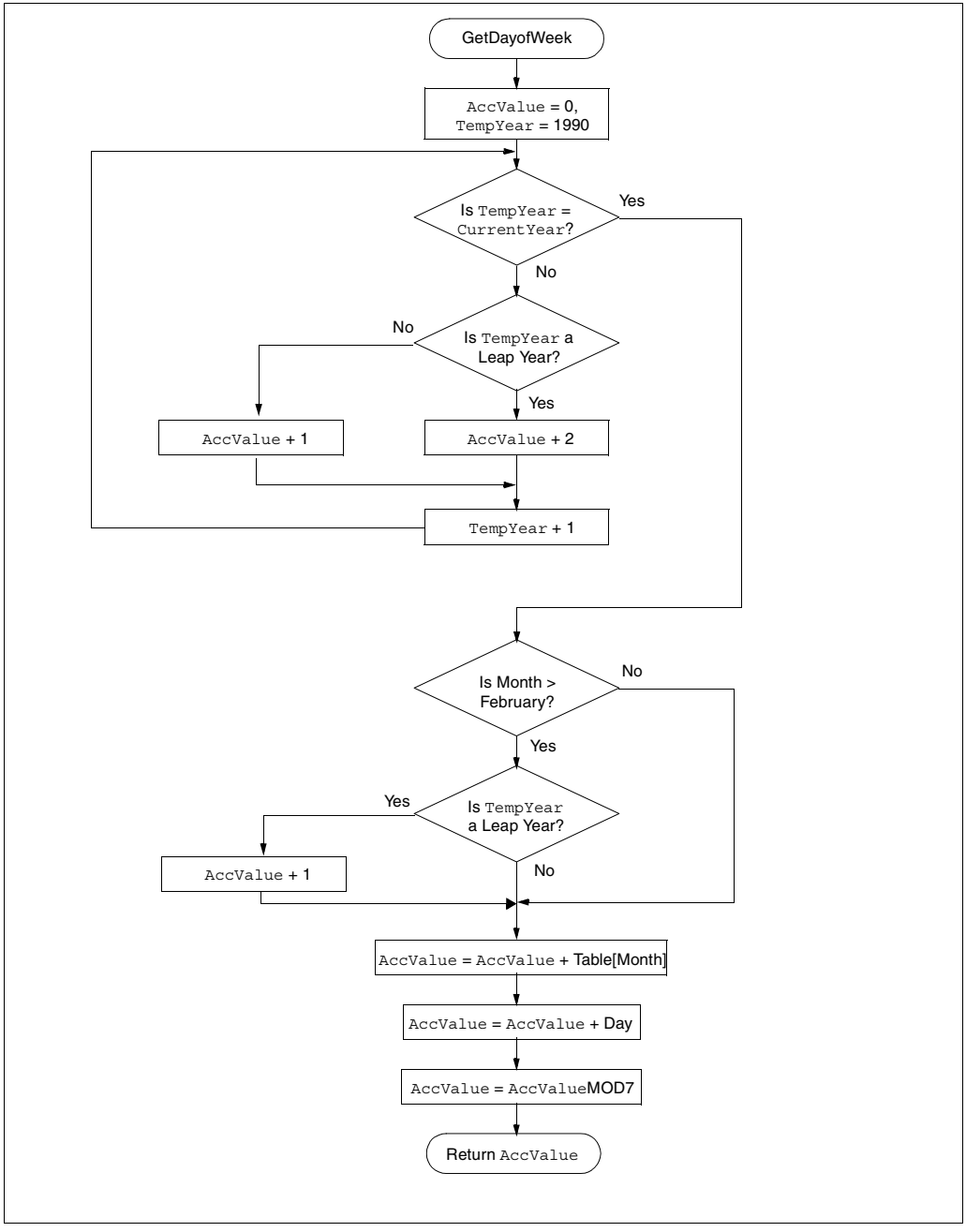
The third step involves adding the day given to the `AccValue`. The `AccValue` is then divided by 7, and the remainder gives the result corresponding to the day of week.

Figure 1 is a flowchart of the software routine.

The software here will only work if the input given ranges from 1 January 1990 to 31 December 2099.

The software does not check for the use of errorneous dates such as 29 February 1998, or 32 March 1999.

FIGURE 1: APPLICATION FLOWCHART



Please check Microchip's Worldwide Website at www.microchip.com for the latest version of source code.

APPENDIX A: PIC16C54 ASSEMBLY CODE

```
        ListP=16C54
;
;*********************************************************
;
PCL             equ     02h
STATUS          equ     03h
;
#define         Z       STATUS,2
#define         C       STATUS,0
;
YearHi          equ     10h             ;Store the Upper byte of Year
YearLw          equ     11h             ;Store the Lower byte of year
Month           equ     12h             ;Store the Month(1 for January, 2 for February and so on)
Day             equ     13h             ;Store the Day
AccValue        equ     14h
TempA           equ     15h
TempB           equ     16h
TempYearHi      equ     15h
TempYearLw      equ     16h
Lbyte           equ     17h
Hbyte           equ     18h
Ltemp           equ     19h
Htemp           equ     1Ah
Temp            equ     1Bh
;
        org 0x00
;*********************************************************
;   Test program for GetDayofWeek
;   The End Result willbe  stored in AccValue
;*********************************************************
;
main            movlw   19h             ;Set Date as 21 September 1998
                movwf   TempA
                movlw   98h
                movwf   TempB
                movlw   9
                movwf   Month
                movlw   D'21'
                movwf   Day
;
                call    BCDtoBin        ;Convert the 4-digit BCD Year to 16bit int value
;
                movfw   Lbyte
                movwf   YearLw
;
                movfw   Hbyte
                movwf   YearHi
;
                call    GetDayofWeek    ;Calculate Day
;
Loop    gotoLoop
;
;*********************************************************
;   Accumulated Value for Month
;   This routine return the remainder value when
;   number of days summed up and divided by 7
;*********************************************************
;
GetMonthValueaddwfPCL,1
                nop
                retlw   0               ;January
                retlw   3               ;February, %(31/7) = 3
```

```
                retlw   3               ;March, Remaider for (3+28)/7
                retlw   6               ;April, (3+31)/7
                retlw   1               ;May, (6+30)/7
                    retlw   4               ;June, (1+31)/7
                    retlw   6               ;July,%(4+30)/7= 6
                    retlw   2               ;August, %(6+31/7) = 2
                    retlw   5               ;September, %(2+31)/7=4
                    retlw   0               ;October, %(5+30)/7=6
                    retlw   3               ;November, (0+31)/7=3
                    retlw   5               ;December, (3+30)/7=5
;
;*********************************************************
;                   Routine : GetDayofWeek
;*********************************************************
;
GetDayofWeekclrfAccValue
                    call    CheckValidInput
                    btfss   Z               ;Is input Valid ?
                    retlw   08
;
                    movlw   0C6h            ;Set the Temp counter to 1990(Decimal)
                    movwf   TempYearLw
                    movlw   07
                    movwf   TempYearHi
;
GetDay_0            call    CompYear        ;Check is Year > Temp counter
                    btfsc   Z
                    goto    GetDay_1        ;Year = Temp
;
;Year is > Temp Year
;
                    incf    AccValue,1
                    call    IsLeapYear      ;Check is TempYear = Leap Year
                    btfsc   Z               ;Is Leap Year ?
                    incf    AccValue,1      ;Yes !
;
                    call    IncTemp
                    goto    GetDay_0
;
GetDay_1            movlw.3
                    subwf   Month,W         ;Check is Month > February ?
                    btfss   C
                    goto    GetDay_2        ;No !
;
                    call    IsLeapYear      ;Check for Leap year
                    btfss   Z
                    goto    GetDay_2        ;Not Leap Year!
                    incf    AccValue,1
;
GetDay_2            movf    Month,w
                    call    GetMonthValue
                    addwf   AccValue,1      ;Sum the AccValue with Month Value
                    movf    Day,w   ;
                    addwf   AccValue,1      ;Sum the AccValue with Day Value
                    call    Modula_7        ;AccValue%7
                    retlw   0
;
;*********************************************************
;                   Routine: Check Valid Input for 1990-2099
;                   (equivalent to 07C6h - 0833h)
;                   If Input Valid, return Valid = 1
;                   Else Valid = 0
;*********************************************************
;
CheckValidInput     movlw   07h
                    subwf   YearHi,w
```

```
                btfss   C
                goto    NotValid        ;YearHi is < 07h
;
                btfss   Z
                goto    CheckValid_0    ;YearHi not equal to 07h
;
;YearHi is = 07h Check For Year low
;
                movlw   0C6h
                subwf   YearLw,w
                btfss   C
                goto    NotValid        ;YearLw is <90
;
ValidYear       bsf     Z
                retlw   0               ;Year is valid
;
;YearHi is greater 07h, check for YearHi=08h
;
CheckValid_0    movlw   08h
                subwf   YearHi,w
                btfss   Z
                goto    NotValid        ;YearHi is > 20
;
;YearHi is 20, check for YearLw < or = 33
;
                movlw   34h
                subwf   YearLw,w
                btfss   C               ;is YearLw end with Hex value ?
                goto    ValidYear       ;Not a valid value
;
NotValid        bcf     Z
                retlw   0
;
;*********************************************************
;               Routine: Modula_7
;               Register: ACC
;               Output: Remaider of Acc/7
;*********************************************************
;
Modula_7        movlw   .7
                subwf   AccValue,w
                btfss   C
                goto    Modula_70
;
;Contents of Acc > 7
;
                movlw   .7
                subwf   AccValue,1
                goto    Modula_7
;
Modula_70       movf    AccValue,w
                movwf   Temp
                retlw   0
;
;*********************************************************
;               Routine : CompYear
;               ReturnZ=1 if Year > TempYear
;               Else return 0
;*********************************************************
;
CompYear        movf    YearHi,w
                subwf   TempYearHi,w
                btfss   Z
                retlw   0               ;YearHi > TempYearHi
;
                movfw   YearLw
```

```
                subwf   TempYearLw,w
                retlw   0
;
;*******************************************************
;                   Routine: IsLeapYear
;                   Return Z=1 if TempYear is Leap year
;                   Else Return 0
;*******************************************************
;
IsLeapYear          btfsc   TempYearLw,0
                    goto    NotLeapYear
;
                    btfsc   TempYearLw,1
                    goto    NotLeapYear
;
                    bsf     Z
                    retlw   0
;
NotLeapYear         bcf     Z
                    retlw   0
;
;*******************************************************
;                   Routine: IncTemp
;                   Increment The Temporary Counter
;*******************************************************
;
IncTemp             incfsz  TempYearLw,1
                    retlw   0
;
                    incf    TempYearHi,1
                    retlw   0
;
;*******************************************************
;                   Routine : BCDtoBin
;                   This routine convert 4-Digit BCD value(D3D2D1D0)
;                   into 16Bit Binary code
;                   input : 2 digit High Byte is stored in TempA
;                         2 digit Low byte is store in TempB
;                   Output: Hbyte:Lbyte
;                   For more on the BCD to Bin conversion please refer
;                   to AN544
;*******************************************************
;
BCDtoBin            clrf    Htemp
                    clrf    Ltemp
                    clrf    Lbyte
                    clrf    Hbyte
;
                    swapf   TempA,w         ;D3*10
                    call    Mpy10
;
                    movfw   TempA           ;[(D3*10)+D2]*10
                    call    Mpy10
;
                    swapf   TempB,w         ;{[(D3*10)+D2]*10+D1}*10
                    call    Mpy10
;
                    movfw   TempB           ;
                    andlw   0x0f
;
                    addwf   Lbyte,1         ;{[(D3*10)+D2]*10+D1}*10+D4
                    btfsc   C
                    incfH   byte,1
                    retlw   0
;
```

```
;*******************************************************
;               Routine : Mpy10
;               This routine multiply the value store in W register by 10
;               Theory : Let say the input is N,
;               1st,  Store the product of 2*N into Temporary register
;               2nd,  Multiply the value N by 8, (8*N) and store in Hbyte and Lbyte
;               3nd,  Sum up the value obtained in 1st and 2nd steps
;               The whole process is equivalent to 2*N+8*N = N(2+8) = 10*N
;*******************************************************
;
Mpy10           andlw   0x0f
                addwf   Lbyte,1         ;2*N and store the product in temp
                btfsc   C
                incf    Hbyte,1
;
                bcf     C
                rlf     Lbyte,w
                movwf   Ltemp
                rlf     Hbyte,W
                movwf   Htemp
;
                bcf     C               ;8*N
                rlf     Lbyte,1
                rlf     Hbyte,1
;
                bcf     C
                rlf     Lbyte,1
                rlf     Hbyte,1
;
                bcf     C
                rlf     Lbyte,1
                rlf     Hbyte,1
;
                movfw   Ltemp           ;8*N+2*N =10*N
                addwf   Lbyte,1
                movfw   Htemp
                addwf   Hbyte,1
                retlw   0
;
;*******************************************************
;
        END
```

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of source code.

APPENDIX B: C IMPLEMENTATION

```
#include<P17C756.H>

#define OK  1
#define Error8

charGetDayofWeek();
charCheckValidInput(unsigned int);
charIsLeapYear(int);

rom     constunsigned charTable[13]={0,0,3,3,6,1,4,6,2,5,0,3,5};

unsigned intCurrentYear;
unsigned charMonth,Day;

/*********************************************************
*   Test Program for the routine GetDayofWeek()
*********************************************************/
voidmain(){

    charTemp;

    CurrentYear = 1998;/*Date : 21 September 1998*/
    Month    = 9;
    Day         = 21;

    Temp = GetDayofWeek();/*Result stored in Temp*/
    do{
    }while(1);
}

/*****************************************************************
*   GetDayofWeek
*   This routine calculate the Day(Sunday, Monday,...Saturday) of
*   week  when a Date(year, Month, Day) is given.
*   Input :  Year, Month and Day which in this routine is used
*   as global variable.
*   Output Variable :  0 to 6(which correspond to Sunday to Saturdaday
*   respectively) if the input is acceptable, else a value 8 is return
*****************************************************************/
charGetDayofWeek(){

    unsigned intTempYear;
    unsigned charAccValue, Temp;

    if(CheckValidInput(CurrentYear)!=OK)/* Return Error if input not Valid*/
        return Error;

    TempYear = 1990;/*Comparation start with year 1990*/
    AccValue = 0;/*Init AccValue to 0*/


/* If TempYear is a leap year AccValue +2, else AccValue+1 */

    while(TempYear != CurrentYear){
        AccValue++;
        if(IsLeapYear(TempYear))
            AccValue++;
        TempYear++;
    }

    if(Month > 2){
        if(IsLeapYear(TempYear)==1)
```

```
                AccValue++;
    }

    AccValue += Table[Month];
    AccValue += Day;

    AccValue= AccValue%7;

    return(AccValue);
}

/*************************************************************************
*   CheckValidInput
*   Return a '1' if the input is within the required range. Else
*   return a '0'.
*
*   Input Variable : 16Bit Unsigned Int
*   Output Varible  : '1' if input ranges between 1990 & 2099 inclusively
*************************************************************************/
charCheckValidInput(unsigned int  Input){

    if(Input>=1990 && Input<=2099 )
        returnOK;
    else
        return  !OK;

}

/*************************************************************************
*   IsLeapYear
*   Return a '1' if the input is a leap year. Else return a '0'
*
*   Input Variable : 16Bit Unsigned Int
*   Output Varible  : '1' if is a leap year, else '0'.
*************************************************************************/
charIsLeapYear(int Year){

    Year=Year&0x0003;
    if(Year==0)
        return 1;
    else
        return0;
}
/*************************************************************************/
```

NOTES:

Complementary LED Drive

Author: *Jean-Claude Rebic*
Pioneer-Standard

INTRODUCTION

Light Emitting Diodes, or LED's, are discrete components able to produce light when a current passes through them. Most microcontroller designs use one or more LED's. This application highlights the utility of driving multiple LED's with a minimum number of I/O pins. Typically, each I/O drives or sources a single LED. To drive more than one, a high I/O count is required. In order to reduce I/O requirements, LED's are multiplexed in a matrix (as found on a keyboard). The complementary LED drive method proposes to implement even more LEDs while using fewer I/O.

LEDs are polarized and can only operate when current flows from anode to cathode (unlike a switch). We can therefore take advantage of this fact. Table 1 shows the number of possible LEDs with respect to the number of I/O pins required. Fifty-six LEDs can be driven using only 8 pins. The only drawback is that only one LED can be driven at a time.

Typical applications include; games, bargraphs, audio, video, or driving a single seven-segment LED display.

TABLE 1: NUMBER OF LEDS WITH RESPECT TO I/O COUNT

I/O pins	2	3	4	5	6	6	8
LEDs	2	6	12	20	30	42	56

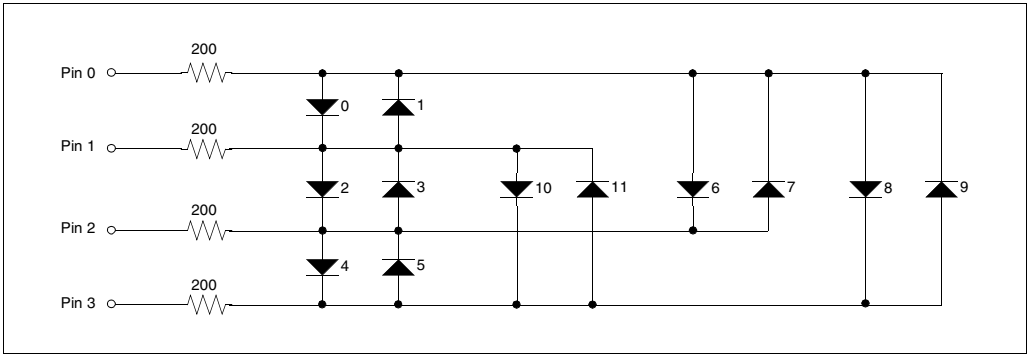
THEORY OF OPERATION

Some microcontrollers available today can sink high current, while others offer a limited number of pins to source high current. Microchip microcontrollers have a very flexible pin structure. When a pin is configured as an input, the input impedance is very high (typically 10 Mohm). When a pin is configured as an output, it can source 20 or 25 mA and sink 25 mA.

To have a better understanding of the application, place two diodes in parallel and reverse the polarities (that is, attach anode to cathode and vice-versa). If you apply 5 volts (with of course a limiting resistor) to one end and ground to another, only one LED will illuminate. The reason is, LED's are polarized and can operate only when current flows from anode to cathode.

Figure 1 gives an example of driving 12 LEDs using only 4 I/Os. To turn an LED on, first configure the appropriate register determining which pins are inputs and which are outputs. Then, write the appropriate voltages on the output pins. Each pin has a 200 ohm resistor to limit the current through the LED's, and since two pins are needed to drive one LED, the resistance is doubled.

FIGURE 1: EXAMPLE OF LED PLACEMENT, RESULTING IN 12 LEDS FOR 4 PINS



There will always be numerous paths for the current to travel between two pins with this technique. Let's take LED 6 for instance (pin 0 and pin 2 configured as outputs, pin 1 and pin 3 configured as inputs; pin 0 is at 5 Vcc and pin 2 is at ground). There are three distinct paths that the current can take:

- Through LED 6
- Through LED 0 in series with LED 2
- Through LED 8 in series with LED 5

Only LED 6 will light up because all three paths have the same voltage drop and all LED's in the series do not have enough of a voltage drop to drive any current.

SPECIAL CONSIDERATIONS

The Complementary LED Drive technique will not work with an open collector output (for example pin RA4 on the PIC16CXX family). Care should be taken when sharing a port with other I/O functions, use a shadow register as a port buffer. Do all operations on the shadow register and write this buffer to the port. It is possible to drive more than one LED at a time, but care must be given in the design. For example, in Figure 1, LEDs 0 and 8 will work if pin 0 (Vcc), pin 1 (Gnd) and pin3 (Gnd) are outputs and pin 2 is an input.

MULTIPLE LEDs AT THE SAME TIME

Trying to turn on more than one LED at a time is a recurrent problem since the Complementary LED Drive technique only allows one LED at a time to be driven. The solution is to have a duty cycle scheme where each LED is turned on sequentially (4 LED's produce a 25% duty cycle). However, there is concern that this process will diminish the brightness level.

Normally, as we increase current flow through an LED, it's brightness increases until it reaches a point where the brightness will actually decrease. This is due to the anode-cathode junction overheating. By running short pulses through the LED at a higher current, we are able to minimize the overheating, and the peak luminosity increases (phenomenon used in GaAsP lasers). For instance, a 10 mA LED has the same intensity to a photometer as a 40 mA pulsed LED with a 25% duty cycle. Both instances produce the same luminosity when measuring the luminosity with a photometer.

Fortunately, the human eye doesn't act as a photometer. It can only combine the average brightness and peak brightness. Our earlier 40 mA example will therefore appear brighter than the 10 mA LED. To increase the current at the maximum rated value of the Microchip microcontroller, use the 25 mA sink/source capability. This pulsing technique is quite useful in battery applications. By pulsing a higher current with a smaller duty cycle, the visual brightness is maintained while consuming less power.

Certain precautions must be taken to use the pulsating technique. First, make sure the LED junction does not overheat, and second, do not dissipate more than the average maximum rated power of the LED.

To learn more about the LED properties in a multiplexed environment, please refer to Siemens Optoelectronics Data Book 1995-1996, Multiplexing LED Displays, Appnote3, p.11-10.

SOFTWARE

As complex as the hardware appears, the software is quite straight forward. Just clear all I/Os associated with the LEDs to remove all glitches. Then load the offset into the accumulator and call a table that configures the I/O TRIS register. Remember that pins configured as outputs will either source (anode of the selected LED) or sink (cathode of the selected LED) current, and all other pins will be configured as inputs. At this point, use the same offset to call a table with the appropriate voltages.

The code is a simple subroutine written for a PIC16C54. Figure 1 is located on PORTA, and a 200 ohm resistor is added for each pin.

CONCLUSION

The Complementary LED Drive will help minimize the number of pins required to drive LEDs in your design, thereby taking advantage of Microchip Technology's smaller 8-pin families.

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of source code.

APPENDIX A: SOFTWARE LISTING

```
Output_Led_
     clrf      PORTA           ; Clear port all to 0
     movf      Led_Value,w     ; Read LED pointer
     call      Table_Tris_     ; Configure i/o direction
     trisa                     ; Write to tris register
     movf      Led_Value,w     ; Read LED pointer
     call      Table_Io_       ; Call table
     movwf     PORTA           ; Write to port
     retlw     0

Table_Io_
     addwf     PCL,f
     retlw     b'00100000'     ; Led 0
     retlw     b'00000010'     ; Led 1
     retlw     b'00100000'     ; Led 2
     retlw     b'00000001'     ; Led 3
     retlw     b'00000010'     ; Led 4
     retlw     b'01000000'     ; Led 5
     retlw     b'00000001'     ; Led 6
     retlw     b'00100000'     ; Led 7
     retlw     b'00000010'     ; Led 8
     retlw     b'00100000'     ; Led 9
     retlw     b'00000001'     ; Led 10
     retlw     b'00000010'     ; Led 11
;
Table_Tris_
     addwf     PCL,f
     retlw     b'01000101'     ; Led 0
     retlw     b'01000101'     ; Led 1
     retlw     b'00000111'     ; Led 2
     retlw     b'01000110'     ; Led 3
     retlw     b'00100101'     ; Led 4
     retlw     b'00100101'     ; Led 5
     retlw     b'00100110'     ; Led 6
     retlw     b'01000101'     ; Led 7
     retlw     b'01000101'     ; Led 8
     retlw     b'00000111'     ; Led 9
     retlw     b'01000110'     ; Led 10
     retlw     b'00100101'     ; Led 11
```

NOTES:

Using the PIC16F877 To Develop Code For PIC16CXXX Devices

Author: *Stan D'Souza, Rodger Richey*
Microchip Technology Inc.

INTRODUCTION

With the release of the FLASH-based PIC16F87X family, Microchip Technology has completed the circle on product technology. Microchip is now in the unique position of offering FLASH, OTP or ROM-based versions of devices with similar feature sets. Customers now have the most flexible position to select their choice of technology and easily migrate from one technology to another, reaping the benefits and cost structures which best suit their needs.

The PIC16F877 FLASH-based PICmicro® introduced by Microchip has the unique distinction of being a superset part with reprogrammable program memory for most PIC16CXXX products. The FLASH program memory, coupled with a built-in, In-Circuit Debugger (ICD), allows the development of a simple, low-cost emulator or ROMulator for most of the PIC16CXXX products.

A typical design scenario starts with the customer developing the initial prototype using a FLASH product. A limited production run using FLASH allows the customer to verify the operation of the production line without sacrificing product. Full production then starts with either FLASH or OTP, depending on how well the software has been validated. When the software has a proven track record of robustness, the customer can migrate to a lower cost ROM. This technical brief will highlight the design considerations associated with such a strategy.

The original goal of the PIC16F87X devices was to maintain compatibility with the existing PIC16C6X/7X devices, such that any code written for a PIC16C6X/7X device will run on a PIC16F87X without any modifications. It is so compatible that any HEX file generated for a PIC16C6X/7X device can be programmed into the PIC16F87X device, as long as the configuration bits are set correctly. Developing an application for an OTP or ROM device using the PIC16F87X requires some care and minor device differences must be taken into account.

The PIC16F87X devices have some features that are not available or modified from the PIC16CXXX devices. When developing code, use the data sheet for the target device and refer to the exceptions listed in the next section for each device. Following these suggestions should minimize the number of problems that arise. The features not found on the OTP/ROM PIC16CXXX microcontrollers are:

- FLASH Program Memory with read and write capabilities
- EEPROM Data Memory
- Master SSP (MSSP)

The peripherals on the PIC16F87X devices that are modified versions of the PIC16CXXX device, but code compatible, are:

- 10-bit A/D converter
- 9-bit Addressable USART

Certain resources are consumed when the MPLAB™-ICD In-Circuit Debugger is used to develop code. These resources are released when the debugger is not used. The resources include:

- Last 256 words of program memory: IF00h to IFFh
- Program memory location 0000h must be a `NOP` instruction
- Pins RB6 and RB7: CLK and Data for ICD
- $\overline{MCLR}/V_{PP}$ used for programming @ 13V
- Data memory locations 70h, 1EBh – 1EFh
- 1 Level of stack

PART-TO-PART DIFFERENCES

When using the PIC16F87X device to develop code for an application that will eventually use a ROM or OTP device, the designer should not only keep in mind the discrepancies between the devices, but also the errata for both devices. A limitation on one or both devices may require the designer to use different workarounds to make the application work. All device errata can be found on the Microchip website (www.microchip.com).

The following paragraphs explain the differences or what's missing on the target device when compared to the PIC16F87X device. Some differences, such as program and data memory sizes, need to be monitored as development progresses to ensure that the limits are not exceeded for the target device. Obviously, none of the advanced features of the PIC16F87X should be used, such as Data EEPROM memory or Master Mode I^2C™. Appendix A shows tables comparing the OTP/ROM PIC16CXXX to the FLASH equivalent.

Data Memory

All of the PIC16CXXX devices, except the PIC16C66/67/76/77, have only two banks of data memory. The method of switching between the two banks is to clear the RP0 bit (STATUS<5>) to access registers in Bank 0 and set the bit to access registers in Bank 1. Any indirect addressing uses only the FSR special function register (SFR) to access any register in Bank 0 or Bank 1. However, the PIC16F87X devices and the PIC16C66/67/76/77 all have four banks of data memory. The RP1:RP0 bits (STATUS<5,6>) now select the bank for direct data memory addressing operations. When performing indirect data memory accesses on a device with four banks, the IRP bit (STATUS<7>) is used to indicate the upper or lower two banks. Since the FSR can access 256 registers or one register in either the two lower or upper banks, only one additional bit is required to indicate which set of registers to operate on. The following table shows the bank decoding of RP1 and RP0

TABLE 1: BANK CONTROL

		RPO	
		0	1
RP1	0	Bank 0	Bank 1
	1	Bank 2	Bank 3

When writing code on the PIC16F87X devices for a target PIC16CXXX device, simply maintain the IRP and RP1 bits as '0's. This requirement follows the recommendations in the target data sheet to maintain these bits clear for devices that have only two banks of data memory.

SSP Module

The next difference is the SSP Module that contains the SPI™ and I^2C peripherals. There are several variations of the SSP Module. Microchip currently has three versions of the SSP Module:

- BSSP
  - 2-mode SPI
  - Slave I^2C with Start & Stop bit detection
- SSP
  - 4-mode SPI with Microwire®
  - Slave I^2C with Start & Stop bit detection
- MSSP
  - 4-mode SPI with Microwire
  - Slave I^2C
  - Master Mode I^2C

Of all the differences between the three modules, the revision of the 2-mode SPI module to 4-modes is the biggest backward compatibility issue. These changes include redefining the functionality of the CKP (SSPCON<4>), adding the CKE bit (SSPSTAT<6>) and adding the SMP bit (SSPSTAT<7>). In the BSSP module, the CKP bit controlled both the idle state of the clock and the clock edge that data is transmitted on as shown below:

- Idle state for clock is high, transmit happens on falling edge, receive on rising edge
- Idle state for clock is low, transmit happens on rising edge, receive on falling edge

In the new 4-mode SPI module, the CKP bit controls the idle state of the clock, the CKE bit controls the edge that data is transmitted and the SMP bit controls where the incoming data is sampled (middle or end of bit time). The SMP bit sets the SPI module for regular SPI operation or Microwire operation. The table below shows this graphically.

TABLE 2: SPI CONFIGURATION

		CKE	
		0	1
CKP	0	low, falling	low, rising
	1	high, rising	high, falling

To properly configure the PIC16F87X devices to be compatible with the non-FLASH devices, you must set the bits to the following states:

- For BSSP operation when CKP = '0', set CKP = '0', CKE = '1', SMP = '0'
- For BSSP operation when CKP = '1', set CKP = '1', CKE = '1', SMP = '0'

Even though the PIC16F87X devices add Master Mode I^2C to the SSP module, problems will not be encountered as long as code is written using the target device data sheet. This part of the SSP module was designed to be 100% compatible with the existing BSSP and SSP modules.

A/D Module

Although the 10-bit A/D converter is 100% compatible to the 8-bit A/D, it does have a 10-bit result. These 10 bits are spread across the two registers, ADRESH and ADRESL, either left or right justified. Any code written for the PIC16F87X 10-bit A/D converter will be compatible as long as the result is left justified (default state) and the code uses ADRESH as the 8-bit result. The ADRESH register is in the same place in the Data Memory map as the ADRES register of the 8-bit A/D. When changing over to the target OTP or ROM device, you simply need to change ADRESH to ADRES.

USART

The addressable USART also has only one additional bit, ADDEN (RCSTA<3>). The default state of this bit is 0, which turns off 9-bit addressing and therefore remains 100% compatible to the non-addressable USART. Here again, problems should not be encountered as long as the target data sheet is used to write code.

Brown-out Reset

On a device that has Brown-out Reset, only the BOR bit (PCON<0>) was added. In most cases, this bit is not used and can be ignored. The main consideration when writing code on the PIC16F87X device for a device that does not have Brown-out Reset is to make sure that this function is disabled in the configuration bits.

Extra SFRs

The PIC16F87X devices will have some SFRs that are not present in other devices. The code should refrain from using these locations, otherwise unexpected results may occur. As long as the code is written using the target device data sheet, problems should not be encountered.

CONCLUSION

The MPLAB-ICD In-Circuit Debugger makes an ideal low-cost development tool for not only the PIC16F87X devices but also for the PIC16C6X/7X devices as well. Using the guidelines presented in this technical brief, a designer should be able to design/debug an application using the FLASH devices and then switch to the lower cost ROM or OTP equivalents for production.

Comparison tables by device family follow for review of feature differences.

APPENDIX A: DEVICE COMPARISON TABLES

PIC16C62A/62B/CR62: 2KW, 28-pin, Without Analog					FLASH Equivalent
Device		**PIC16C62A**	**PIC16C62B**	**PIC16CR62**	**PIC16F873**
Program Memory		2KW	2KW	2KW	4KW
Data Memory	**Bytes**	128	128	128	192
	Banks	0,1	0,1	0,1	0,1,2,3
	Shared	—	—	—	16 bytes (F0h-FFh)
Data EEPROM		128	128	128	128
Additional Features		—	—	—	10-bit A/D Addressable RT CCP2
Additional SFRs		—	—	—	PIR2, PIE2, CCPR2L, CCPR2H, CCP2CON, RXSTA, TXSTA, SPBRG, ADCON0, ADCON1, ADRESH, ADRESL, SSPCON2, TXREG, RCREG
Bit Differences	**IRP**	Reserved	Reserved	Reserved	Selects upper/lower bank
	RP1	Reserved	Reserved	Reserved	Selects upper/lower bank
	SMP	—	Yes	—	Microwire/slew rate control
	CKE	—	Yes	—	Clock edge/levels select
	CKP	Yes	Yes	Yes	Clock idle/stretching control

PIC16C63A/CR63: 4KW, 28-pin, Without Analog				FLASH Equivalent
Device		**PIC16C63A**	**PIC16CR63**	**PIC16F873**
Program Memory		4KW	4KW	4KW
Data Memory	**Bytes**	192	192	192
	Banks	0,1	0,1	0,1,2,3
	Shared	—	—	16 bytes (F0h-FFh)
Data EEPROM		—	—	128
Additional Features		Brown-out Reset	Brown-out Reset	10-bit A/D Brown-out Reset
Additional SFRs		—	—	ADCON0, ADCON1, ADRESH, ADRESL, SSPCON2
Bit Differences	**IRP**	Reserved	Reserved	Selects upper/lower bank
	RP1	Reserved	Reserved	Selects upper/lower bank
	SMP	Yes	—	Microwire/slew rate control
	CKE	Yes	—	Clock edge/levels select
	CKP	Yes	Yes	Clock idle/stretching control
	BOR	Yes	Yes	Brown-out Reset status
	ADDEN	—	—	Selects 9-bit addressing

PIC16C64A/CR64: 2KW, 40-pin, Without Analog				FLASH Equivalent
Device		**PIC16C64A**	**PIC16CR64**	**PIC16F874**
Program Memory		2KW	2KW	4KW
Data Memory	**Bytes**	128	128	192
	Banks	0,1	0,1	0,1,2,3
	Shared	—	—	16 bytes (F0h-FFh)
Data EEPROM		—	128	128
Additional Features		Brown-out Reset	—	10-bit A/D Addressable USART CCP2 Brown-out Reset
Additional SFRs		—	—	PIR2, PIE2, CCPR2L, CCPR2H, CCP2CON, RXSTA, TXSTA, SPBRG, ADCON0, ADCON1, ADRESH, ADRESL, SSPCON2, TXREG, RCREG
Bit Differences	**IRP**	Reserved	Reserved	Selects upper/lower bank
	RP1	Reserved	Reserved	Selects upper/lower bank
	SMP	—	—	Microwire/slew rate control
	CKE	—	—	Clock edge/levels select
	CKP	Yes	Yes	Clock idle/stretching control
	BOR	Yes	Yes	Brown-out Reset status

PIC16C65A/65B/CR65: 4KW, 40-pin, Without Analog					FLASH Equivalent
Device		**PIC16C65A**	**PIC16C65B**	**PIC16CR65**	**PIC16F874**
Program Memory		4KW	4KW	4KW	4KW
Data Memory	**Bytes**	192	192	192	192
	Banks	0,1	0,1	0,1	0,1,2,3
	Shared	—	—	—	16 bytes (F0h-FFh)
Data EEPROM		—	—	128	128
Additional Features		Brown-out Reset	Brown-out Reset	Brown-out Reset	Brown-out Reset
Additional SFRs		—	—	—	ADCON0, ADCON1, ADRESH, ADRESL, SSPCON2
Bit Differences	**IRP**	Reserved	Reserved	Reserved	Selects upper/lower bank
	RP1	Reserved	Reserved	Reserved	Selects upper/lower bank
	SMP	Yes	Yes	—	Microwire/slew rate
	CKE	Yes	Yes	—	Clock edge/levels select
	CKP	Yes	Yes	Yes	Clock idle/stretching
	BOR	Yes	Yes	Yes	Brown-out Reset status
	ADDEN	—	—	—	Selects 9-bit addressing

PIC16C66/67: 8KW, 28/40-pin, Without Analog				FLASH Equivalent
Device		**PIC16C66**	**PIC16C67**	**PIC16F876/877**
Program Memory		8KW	8KW	8KW
Data Memory	**Bytes**	368	368	368
	Banks	0,1,2,3	0,1,2,3	0,1,2,3
	Shared	16 bytes (F0h-FFh)	16 bytes (F0h-FFh)	16 bytes (F0h-FFh)
Data EEPROM		—	—	128
Additional Features		—	—	10-bit A/D
Additional SFRs		—	—	ADCON0, ADCON1, ADRESH, ADRESL, SSPCON2
Bit Differences	**ADDEN**	ADDEN	—	Selects 9-bit addressing

PIC16C72/72A/CR72: 2KW, 28-pin, With Analog					FLASH Equivalent
Device		**PIC16C72**	**PIC16C72A**	**PIC16CR72**	**PIC16F873**
Program Memory		2KW	2KW	2KW	4KW
Data Memory	**Bytes**	128	128	128	192
	Banks	0,1	0,1	0,1	0,1,2,3
	Shared	—	—	—	16 bytes (F0h-FFh)
Data EEPROM		—	—	—	128
Additional Features		—	—	—	Addressable USART CCP2
Additional SFRs		—	—	—	PIR2, PIE2, CCPR2L, CCPR2H, CCP2CON, RXSTA, TXSTA, SPBRG, SSPCON2, TXREG, RCREG, ADRESL
Bit Differences	**IRP**	Reserved	Reserved	Reserved	Selects upper/lower bank
	RP1	Reserved	Reserved	Reserved	Selects upper/lower bank
	SMP	—	Yes	Yes	Microwire/slew rate control
	CKE	—	Yes	Yes	Clock edge/levels select
	CKP	Yes	Yes	Yes	Clock idle/stretching control
Register Names	**ADRESH**	ADRESH	ADRES	ADRES	ADRESH in F873 same as ADRES

PIC16C73A/73B: 4KW, 28-pin, With Analog				FLASH Equivalent
Device		**PIC16C73A**	**PIC16C73B**	**PIC16F873**
Program Memory		4KW	4KW	4KW
Data Memory	**Bytes**	192	192	192
	Banks	0,1	0,1	0,1,2,3
	Shared	—	—	16 bytes (F0h-FFh)
Data EEPROM		—	—	128
Additional Features		Brown-out Reset	Brown-out Reset	Brown-out Reset
Additional SFRs		—	—	ADRESL,SSPCON2
Bit Differences	**IRP**	Reserved	Reserved	Selects upper/lower bank
	RP1	Reserved	Reserved	Selects upper/lower bank
	SMP	Yes	Yes	Microwire/slew rate control
	CKE	Yes	Yes	Clock edge/levels select
	CKP	Yes	Yes	Clock idle/stretching control
	BOR	Yes	Yes	Brown-out Reset status
	ADDEN	—	—	Selects 9-bit addressing
Register Names	**ADRESH**	ADRES	ADRES	ADRESH in F873 same as ADRES

PIC16C74A/74B: 4KW, 40-pin, With Analog				FLASH Equivalent
Device		**PIC16C74A**	**PIC16C74B**	**PIC16F874**
Program Memory		4KW	4KW	4KW
Data Memory	**Bytes**	192	192	192
	Banks	0,1	0,1	0,1,2,3
	Shared	—	—	16 bytes (F0h-FFh)
Data EEPROM		—	—	128
Additional Features		Brown-out Reset	Brown-out Reset	Brown-out Reset
Additional SFRs		—	—	ADRESL,SSPCON2
Bit Differences	**IRP**	Reserved	Reserved	Selects upper/lower bank
	RP1	Reserved	Reserved	Selects upper/lower bank
	SMP	Yes	Yes	Microwire/slew rate control
	CKE	Yes	Yes	Clock edge/levels select
	CKP	Yes	Yes	Clock idle/stretching control
	BOR	Yes	Yes	Brown-out Reset status
	ADDEN	—	—	Selects 9-bit addressing
Register Names	**ADRESH**	ADRES	ADRES	ADRESH in F874 same as ADRES

PIC16C76/77: 8KW, 28/40-pin, Without Analog				FLASH Equivalent
Device		**PIC16C76**	**PIC16C77**	**PIC16F876/877**
Program Memory		8KW	8KW	8KW
Data Memory	**Bytes**	368	368	368
	Banks	0,1,2,3	0,1,2,3	0,1,2,3
	Shared	16 bytes (F0h-FFh)	16 bytes (F0h-FFh)	16 bytes (F0h-FFh)
Data EEPROM		—	—	128
Additional Features		—	—	10-bit A/D
Additional SFRs		—	—	ADRESL, SSPCON2
Bit Differences	**ADDEN**	—	—	Selects 9-bit addressing
Register Names	**ADRESH**	ADRES	ADRES	ADRESH in F876/77 same as ADRES

SECTION 3
SECURE DATA PRODUCT APPLICATION NOTES AND TECHNICAL BRIEFS

AN650

Designing a Transponder Coil for the HCS410

Authors: Mike Sonnabend, Jan van Niekerk
Microchip Technology Inc.

OVERVIEW

This application note explains the design of transponder coils. An Excel spreadsheet is used to automate the update of values, depending on the specified parameters. The spreadsheet file name is `transpnd.xls`. A zip file containing this spreadsheet and a copy of this application note can be downloaded from Microchip's web site at *www.microchip.com*.

The basic approach is to choose the transponder coil external dimensions, since volume will usually be the primary constraint for a coil as it will need to fit into a keyfob, credit card or other small volume. Secondly, properties of the core, coil windings as well as the equivalent load placed across the coil are entered. This fixes the Initial Coil Specification.

Once the initial coil is built, measurements are made on this coil to determine the coil quality factor. These measurements are used to calculate the Optimum Coil Specification for a second coil.

The authors welcome feedback, comments, questions and errata via e-mail.
mike.sonnabend.@microchip.com
jan.van.niekerk@microchip.com

SPREADSHEET FEATURES

The spreadsheet is split into three worksheets. The first worksheet concerns the initial coil specification. The inputs to this worksheet are the:

- coil external dimensions
- core effective relative permeability
- wire resistivity
- coil-packing factor
- transponder resonant frequency
- equivalent load that the HCS410 presents to the resonant circuit.

The worksheet output gives the minimum number of turns that the coil is required to have. The number of turns together with coil dimensions fix the coil inductance, wire resistance, resonating capacitor and wire diameter.

The second worksheet enables the user to change the number of turns from what is suggested in the initial coil specification in order to use a standard value resonating capacitor.

The third worksheet requires a quality (Q) factor measurement to be made on the initial coil when it is optimally resonated. The two measured voltage values are the only inputs required to determine the number of turns for the optimal coil. Once again, the optimal number of turns together with the same coil dimensions fix the coil inductance, wire resistance, resonating capacitor and wire diameter. The second worksheet can also be used to change the number of turns from what is suggested in the Optimum Coil Specification in order to use a standard value resonating capacitor.

INTRODUCTION

Overview of Inductive Communication

Communication between a KEELOQ® transponder and a base station occurs via magnetic coupling between the transponder coil and base station coil. The base station coil forms part of a series Resistance Inductor Capacitor (RLC) circuit. The base station communicates to the transponder by switching the 125kHz signal to the series RLC circuit on and off. Thus, the base station magnetic field is switched on and off.

The transponder coil is connected in parallel with a resonating capacitor (125kHz) and a KEELOQ HCS410 transponder integrated circuit. When the transponder is brought into the base station magnetic field, it magnetically couples with this field and draws energy from it. This loading effect can be observed as a decrease in voltage across the base station resonating capacitor.

The KEELOQ transponder communicates to the base station by "shorting out" its parallel LC circuit. This detunes the transponder and removes the load, which is observed as an increase in voltage across the base station resonating capacitor. The base station capacitor voltage is the input to the base station AM-demodulator circuit. The demodulator extracts the transponder data for further processing by the base station software.

Using the Spread Sheet

Color Coding

The spreadsheet is color coded as shown in the table below.

Color	Meaning
Green	User input. The default values correspond to the HCS410 EV kit transponder coil.
Red	Output.
Gray	System defined.

Units

The units in the spreadsheet have been made SI units. Below is a table with some of the most common conversions that the user may come across.

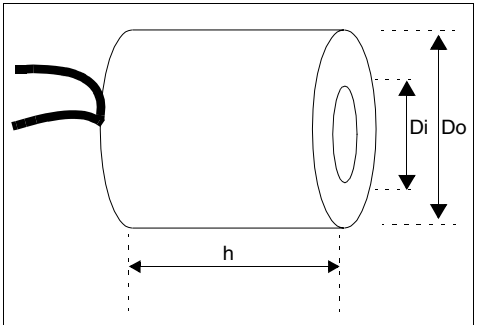
Conversion from:	Operation
Inches (in) to meters (m)	* .0254
Inches (in) to centimeters (cm)	* 2.54
Inches (in) to millimeters (m)	* 25.4
Centimeters (cm) to meters (m)	* 0.01
Millimeters (mm) to meters (m)	* 0.001
Farads (F) to pico farads (nF)	* 1e-9
Henry (H) to micro henry (µH)	* 1e-6

WORKSHEET 1: INITIAL COIL SPECIFICATION

Data Required

The first step is to decide on the dimensions that the coil should have. The dimensions required by the spreadsheet are shown in Figure 1 below.

FIGURE 1: COIL DIMENSIONS



Coil Dimensions			
Input	**Units**	**Typ. Value**	**Description**
h	mm	9	Height of coil along its axis.
Do	mm	2.3	Outside diameter of coil.
Di	mm	1.5	Inside diameter of coil. Also core outside diameter if core is used.

Ferrite Core Usage

The next step is to decide if a ferrite core (also called a ferrite slug) is going to be used or not. If an air core is used, then the relative permeability is 1.

There are advantages and disadvantages to using a ferrite core. The advantage is that the coil can have a larger inductance for a given volume. The disadvantage is that the effective permeability can be very sensitive to the core mechanical dimensions.

One method used to get the exact inductance for a coil wound onto a ferrite core is to have sets of samples built up, each with a different numbers of turns wound onto the cores. Measurements for these coils and interpolation will yield the correct number of turns for the ferrite core. Alternatively, some manufacturers will wind the coil onto the ferrite core to a specified inductance.

Ferrite core manufacturers usually publish curves of slug effective permeability vs. slug length divided by slug diameter. There is a large difference between the ferrite material permeability (typically 2300) and the effective permeability of a slug (typically 23). The effective permeability must be used in the spreadsheet.

Core Permeability			
Input	Units	Typ. Value	Description
μr		23.5	Effective relative permeability. This is the ratio of magnetic field strength inside the coil with the core in place, to the magnetic field strength if an air core replaces the core.

Wire Resistivity

The default values in the spreadsheet assume annealed copper wire. The wire resistivity need not be changed unless a different wire material is used. The packing factor can be left at 0.5 if the coil is tightly wound with wire that has a circular cross section.

Wire Resistivity			
Input	Units	Typ. Value	Description
ρ	ohm-m	1.72E-08	Coil wire resistivity at 20°C. Resistivity for annealed copper wire is used. If the coil uses another type of wire, then the corresponding resistivity would have to be used.
K		0.5	Packing factor. This compensates for copper area lost due to wire shape that is round and not square as well as wire insulation. If the coil is wound by hand, then the space factor of less than 0.5 may have to be chosen to compensate for wasted space.

Magnetic Field Operating Frequency

The magnetic field is generated by the base-station and the frequency is set at the base-station. The transponder coil operates at the same frequency and should match the base station magnetic field operating frequency i.e. 125kHz.

Field Operating Frequency			
Input	Units	Typ. Value	Description
F	kHz	125	Coil operating frequency (resonance).

HCS410 Load

The equivalent average load that the HCS410 presents to the transponder resonant circuit can change with the HCS410 configuration i.e. this value will be higher if auto damping is not selected. The average load is in the order of mega ohms when the HCS410 is battery powered.

The HCS410 pool capacitor will average out the resistance of the coil except during transponder to base communication and during auto damping when the HCS410 “shorts” out the coil.

One method to measure the average load is to use a DM303003 HCS410 Evaluation Kit with a transponder that is perfectly resonated. The base-station and transponder are to be programmed in the same way as for the final application. The transponder is placed in the field. The voltage is measured across the coil using a high impedance oscilloscope probe. The coil voltage and exact position of the coil is noted where the transponder just stops working.

The HCS410 is then replaced with a variable resistor. Keeping the coil in exactly the same position noted above, the variable resistor is adjusted until the voltage is exactly the same. This resistor value is the value to be used as equivalent HCS410 load. The value will be in the order of 100k ohm if no battery is used.

HCS410 Load			
Input	Units	Typ. Value	Description
RP	Ω	60000	This is an equivalent average load that the HCS410 presents to the transponder coil.

Intermediate Calculations

The variables used to calculate the initial coil are given in the table below.

Initial Coil Specifications - Variables			
Input	**Units**	**Typical Value**	**Description**
ωr	rad/sec	785398.1634	Transmission frequency in radians per second.
A	sq. mm	36	Area for packing wire into.
RONE	ohms	5.70374E-5	Coil resistance of one turn which occupies the total volume.
LONE	Henry	7.36485E-9	Coil inductance of one turn which occupies the total volume.

Output Data

The output data for the initial coil matches the HCS410 equivalent load to the wire resistance and results in minimum number of turns.

Initial Coil Specifications - Output Data			
Input	**Units**	**Typical Value**	**Description**
NMIN	Turns	319.8168897	This is the minimum number of turns that the coil should have to match the HCS410 local resistance.
LMIN	µH	753.2980565	The minimum coil inductance, since inductance increases with number of turns.
RMIN	Ω	5.833943331	Minimum wire resistance, since resistance increases with number of turns.
CMAX	pF	2152.055119	This is the maximum capacitance for the resonating capacitor. The product of LMIN and CMAX is a constant determined from resonant frequency.
DMAX	mm	0.084652661	This is the maximum wire diameter, as fitting more turns into a constant volume requires thinner wire to be used.

WORKSHEET 2: USER ENTERS NUMBER OF TURNS

Data Required

The number of turns as suggested for NMIN or NOPT can be entered as the number of turns. This number can be changed slightly until the output results are as desired i.e. to use a standard value resonating capacitor. Note that too large a change in N will result in a non-optimal coil.

User Enters Number of Turns			
Input	**Units**	**Typ. Value**	**Description**
N	Turns	350	The user is free to select a number of turns that the coil should have. This is useful in order to match the inductance to a standard value capacitor

Output Data

The worksheet output includes the resonating capacitor value which should be a standard value component.

User Enters Number of Turns			
Input	**Units**	**Typical Value**	**Description**
L	uH	902.194437	Coil inductance for number of turns entered by user.
RWIRE	Ω	6.987076595	Wire resistance for number of turns entered by user.
CRES	pF	1796.88421	Resonating capacitor to resonate coil with number of turns entered by user.
DWIRE	mm	0.080920264	Wire diameter; choose closest available wire diameter.

WORKSHEET 3: QUALITY FACTOR MEASUREMENT

The coil shape factor M is obtained from measurements made on the initial coil design. This can only be done after the initial coil has been designed and built. This factor is calculated as follows:

1. Place the coil into the base-station magnetic field.
2. Resonate the coil with a capacitor placed in parallel with the coil, which is the same type (dissipation factor) as the capacitor which will be used with the optimally designed coil.
3. Measures the voltage (VCAP) across the coil using a high impedance oscilloscope probe.
4. Disconnect the capacitor while keeping the resonant circuit in EXACTLY the same place with respect to the magnetic field.
5. Now measure the voltage (VNO_CAP) across the coil using the high impedance oscilloscope probe.

Data Required

The two voltage measurements from the quality factor measurement are the only data required.

Q Factor Measurement			
Input	**Units**	**Typ. Value**	**Description**
VCAP	V	46.25	Voltage across the initial coil plus capacitor when the coil is resonant.
VNO_CAP	V	20.94	Voltage across initial coil with capacitor disconnected and coil kept in exactly the same place.

Intermediate Calculations

The coil shape factor M is found from Q factor measurements on the first coil design. It can subsequently be used for optimization of further coil designs while keeping coil dimensions constant.

Field			
Input	**Units**	**Typical Value**	**Description**
Q		22.086915	Quality factor calculated from VCAP/VNO_CAP
M		3.591551527	This is a proportionality factor between the internal resistance of the coil observed in Q factor measurements and the DC resistance of the coil wire as measured with a multimeter. This factor is dependent on the coil's physical geometry and thus called the "shape factor" M in this application note.
RINT	Ω	0.000204853	Internal resistance for one turn.

Output Data

The Optimum Resonant Circuit Specification is derived for the case where the shape factor has been calculated from Q-factor measurements made on the initial coil design. The number of turns will always be greater than for the initial coil

Optimum Coil Specification			
Input	**Units**	**Typical Value**	**Description**
NOPT	Turns	6.85.3006229	Optimum number of turns that the coil should have once shape factor M is known.
LOPT	uH	3458.806841	Coil inductance for optimum number of turns.
ROPT	Ohm	26.78685141	Wire resistance for optimum number of turns.
COPT	pF	468.6988932	Resonating capacitor to resonate coil with optimum number of turns.
DOPT	mm	0.057829675	Optimum wire diameter; choose closest available wire diameter.

APPENDIX A: FORMULAS USED IN THE SPREADSHEET

This appendix gives the formulas used in the spreadsheet. All values use metric units.

For a frequency f in Hertz, the radians per second frequency is given by:

$$\omega_r = 2\pi f$$

Area A for packing wire is determined from coil outside diameter Do, coil inside diameter Di and coil axial length h as:

$$A = \frac{(Do - Di)}{2} \times h$$

RONE is the wire resistance of one turn of wire that has resistivity ρ and occupies total available volume adjusted by packing factor K:

$$R_{ONE} = \rho\pi\frac{(Do + Di)}{2AK}$$

LONE the inductance for one turn of wire that occupies the total available volume, wound onto a core of relative permeability μr is given by:

$$L_{ONE} = \frac{\mu_r (Do + Di)^2}{127000(26Do + 36h - 14Di)}$$

Minimum number of turns NMIN, for a load with equivalent resistance RP connected in parallel across the parallel resonant circuit is given by:

$$N_{MIN} = \frac{\sqrt{R_P R_{ONE}}}{\omega_r L_{ONE}}$$

The coil inductance for N turns is given by:

$$L = L_{ONE} N^2$$

The reason for the factor N squared comes from the empirical formula given in reference [1].

Similarly the resistance of the coil wire RWIRE is given by:

$$R_{WIRE} = R_{ONE} N^2$$

The reason for the factor N squared and not just N is that for a constant coil volume, increasing N from 1 turn to N turns increases the resistance as follows:

- Resistance increases by a factor of N due to coil length increasing N times
- Resistance also increases by a further factor of N due to the wire cross sectional area being reduced by a factor of N due to the constant area for the conductors to fit into.

The resonant capacitor C is given by the formula:

$$C = \frac{1}{\omega_r^2 L}$$

The wire diameter DWIRE is given by the formula:

$$D_{WIRE} = 2\sqrt{\frac{AK}{\pi N}}$$

The closest available wire diameter is chosen to wind the coil with.

With coil quality factor measurement giving the voltage VCAP across the coil when it is perfectly resonated with a resonating capacitor, and VNO_CAP when the capacitor is disconnected, the coil quality factor Q is calculated as:

$$Q = \frac{V_{CAP}}{V_{NO\_CAP}}$$

The transponder resonant circuit consists of a coil connected in parallel with a capacitor. The quality factor of this resonant circuit Q is defined as:

$$Q = \frac{\omega L}{R_{TOTAL}}$$

Appendix B; Nature of the internal resistance "RINT" shows that the internal flux resistance for one turn of wire RINT it is related to the wire resistance for one turn of wire RONE by the measured magnetic shape factor M:

$$R_{INT} = R_{ONE} M$$

and that the total coil resistance can be written as:

$$R_{TOTAL} = (R_{ONE} + R_{INT}) N^2$$

Using the three equations above and coil inductance for N turns, the coil shape factor can be written as:

$$M = \frac{\omega L_{ONE}}{Q R_{ONE}} - 1$$

From Appendix E, the optimum number of turns N, for a load with equivalent resistance RP connected in parallel across the parallel resonant circuit is given by:

$$N = \frac{\sqrt{R_P (R_{ONE} + R_{INT})}}{\omega_r L_{ONE}}$$

APPENDIX B: NATURE OF THE INTERNAL RESISTANCE RINT

Induced Electromotive Force

Faraday's law states that the induced electro motive force (EMF) E in a circuit is numerically equal to the rate of change of the flux Φ through it. Expanding this statement for a coil of N turns in which the flux Φ varies at the same rate through each turn gives the induced EMF as:

$$E = -N\frac{d\Phi}{dt}$$

If the coil has a core other than vacuum, then the magnetic flux is increased by a factor equal to the relative permeability μr:

$$E = -\mu_r N\frac{d\Phi}{dt}$$

Letting the magnetic flux be:

$$\Phi = \hat{\Phi}cos(\omega t)$$

gives:

$$E = \mu_r N\omega \hat{\Phi}sin(\omega t)$$

This voltage is the voltage across the coil when no resonating capacitor is used. Since no current flows through the coil, any resistances associated with the coil do not affect E.

Induced Current

If the coil is shorted out with a capacitor that resonates with the coil inductance at the magnetic field frequency, then induced current flows through the coil. The current in turn generates a magnetic field that opposes the magnetic field that creates it.

Using the Biot Savart law it can be shown that the magnetic flux produced by a coil is given by:

$$\Phi = k\,\mu_0\,\mu_r\,NI$$

where:

μ_0 is the permeability of vacuum.

μ_r is the relative permeability of the core.

N is the number of turns on the coil.

I is the RMS current flowing in the coil due to the base station magnetic field.

k is a factor based on the coil dimensions.

Φ is the flux through the coil due to the current I.

The flux is produced by the current is opposite in phase to base-station flux but proportional in magnitude.

The transponder circuit current is thus:

$$I = \frac{\Phi}{k\,\mu_0\,\mu_r\,N}$$

Transponder Resistance

The transponder coil internal resistance RFLUX due to magnetic flux considerations is thus:

$$R_{FLUX} \propto \frac{E}{I}$$

Substituting for E and I gives:

$$R_{FLUX} \propto k\,\mu_0\,\mu_r^{\,2}\,\omega\,N^2$$

It can be seen that RFLUX is independent of the magnetic flux Φ but it is proportional to the coil physical dimensions that determine k:

Using a proportionality factor P:

$$R_{FLUX} = P\,k\,\mu_0\,\mu_r^{\,2}\,\omega N^2$$

letting:

$$R_{INT} = P\,k\,\mu_0\,\mu_r^{\,2}\,\omega$$

then:

$$R_{FLUX} = R_{INT}\,N^2$$

The transponder coil wire resistance for a constant coil volume, is equal to:

$$R_{WIRE} = R_{ONE}\,N^2$$

where:

$$R_{ONE} = \rho\pi\frac{(Do+Di)}{2AK}$$

and:

ρ is the resistivity of coil wire material i.e. copper.

Do is the coil outside diameter.

Di is the coil inside diameter.

A is the total area for packing the wire into.

K is the packing factor i.e. fraction of coil cross section occupied by copper.

The nature of RFLUX is that it is proportional to N squared as is RWIRE. Thus RINT can be set proportional to RONE by using a factor of proportionality M. This will remain true regardless of the number of turns N and the magnetic flux Φ.

$$R_{INT} = MR_{ONE}$$

Thus the total resistance for the transponder can be written as:

$$R_{TOTAL} = R_{WIRE} + R_{FLUX}$$

Substituting for RWIRE and RFLUX:

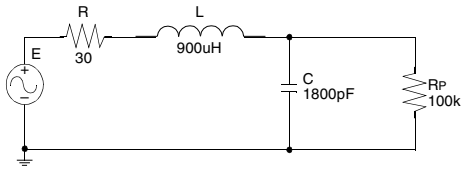
$$R_{TOTAL} = R_{ONE}\,N^2 + R_{INT}\,N^2$$

Substituting for RINT:

$$R_{TOTAL} = R_{ONE}\,N^2 + MR_{ONE}\,N^2$$

APPENDIX C: RESONANT FREQUENCY FOR TRANSPONDER

The equivalent circuit for a transponder is shown below.



E is a voltage source that represents the voltage induced in the coil due to the magnetic field.

R represents the total internal resistance of the coil.

L is the coil inductance.

C is the resonating capacitor capacitance.

Rp is the resistive load presented to the resonant circuit by the HCS410.

The resonant frequency for the above circuit will now be derived.

The impedance seen by E is

$$Z = R + i \times w \times L + \frac{R_P \frac{1}{i \times w \times C}}{R_P + \frac{1}{i \times w \times C}}$$

Manipulating this gives:

$$Z = \frac{(R \times {R_P}^2 \times w^2 \times C^2 + R + i \times w^3 \times L \times {R_P}^2 \times C^2 + i \times w \times L - i \times {R_P}^2 \times w \times C + R_P)}{({R_p}^2 \times w^2 \times C^2 + 1)}$$

At resonance, the impedance is purely resistive, which means that's the imaginary portion of the equation is zero. This gives frequency as:

$$\omega = \sqrt{\frac{1}{LC} + \frac{1}{{R_P}^2 C^2}}$$

It can be seen that the load affects the resonant frequency. If the load is taken off, then the resonant frequency is:

$$\omega = \frac{1}{\sqrt{LC}}$$

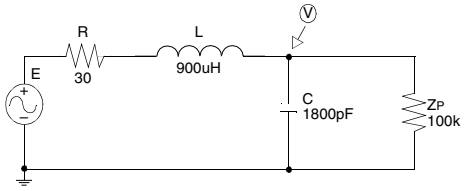
If the load resistance is decreased, the point at which the number inside the square root becomes negative causes the frequency to be imaginary and oscillation stops.

To calculate at which point the load lowers the resonant frequency by x%, using the equations above:

$$R_P = \frac{100}{\sqrt{x}\sqrt{200 - x}} \sqrt{\frac{L}{C}}$$

APPENDIX D: MAXIMUM POWER TRANSFER

It is desirable to match resonant circuit to the load Zp for maximum power transfer.



The voltage across the load ZP is:

$$V = E \times \frac{Z_P \times \frac{1}{i \times w \times C}}{Z_P + \frac{1}{i \times w \times C}} \Bigg/ R + i \times w \times L + \frac{Z_P \frac{1}{i \times w \times C}}{Z_P + \frac{1}{i \times w \times C}}$$

This can be simplified to:

$$V = \frac{i \times Z_p \times E}{(R \times Z_P \times w \times C - i \times R + i \times w^2 \times L \times Z_p \times C + w \times L - i \times Z_p)}$$

The power in the load ZP is given by:

$$P = \frac{V^2}{Z_p}$$

Substituting for V gives:

$$P = -Z_p \times \frac{E^2}{(R \times Z_p \times w \times C - i \times R + i \times w^2 \times L \times Z_p \times C + w \times L - i \times Z_p)^2}$$

To find the maximum power transfer to the load ZP, the expression for power is differentiated with respect to ZP and set to zero. The roots of this equation will give value of ZP for maximum power transfer.

$$\frac{d}{dZ_p} P = 0$$

Solving for ZP gives:

$$Z_P = \frac{-(i \times R - w \times L)}{(R \times w \times C + i \times w^2 \times L \times C - i))}$$

At resonance, assuming that ZP has a negligent effect on frequency:

$$\omega = \frac{1}{\sqrt{LC}}$$

which can be written as:

$$C = \frac{1}{L\omega^2}$$

Substituting for C into the optimum ZP equation gives:

$$Z_P = \frac{(-i \times R + w \times L)}{R} \times w \times L$$

The quality factor Q is defined as:

$$Q = \frac{\omega L}{R}$$

which can be re-written as:

$$R = \frac{\omega L}{Q}$$

Substituting this R into the equation for ZP gives:

$$Z_P = \omega L(Q - i)$$

This shows that for maximum power transfer, the load must have a capacitive reactance equal to (L and a resistance component equal to (LQ. Since the Q for a transponder is always going to be above 10, and since the HCS410 is modeled as a resistive load, maximum power transfer occurs when:

$$Z_P = \omega LQ$$

APPENDIX E: OPTIMUM NUMBER OF TURNS

Have already shown in "APPENDIX D: Maximum Power Transfer" that maximum power transfer occurs when:

$$Z_P = \omega LQ$$

Q is defined as:

$$Q = \frac{\omega L}{R}$$

Substituting for Q in ZP gives:

$$Z_P = \frac{\omega^2 L^2}{R}$$

"Appendix A: Formulas used in the spreadsheet" gives inductance as:

$$L = L_{ONE} N^2$$

"Appendix B: Nature of the Internal Resistance RINT" gives the total resistance in the transponder resonant circuit as:

$$R_{TOTAL} = R_{ONE} N^2 + R_{INT} N^2$$

Substituting the above two equations for L and R in ZP gives:

$$Z_P = \frac{\omega^2 L_{ONE}{}^2 N^4}{R_{ONE} N^2 + R_{INT} N^2}$$

Simplifying:

$$Z_P = \frac{\omega^2 L_{ONE}{}^2 N^2}{R_{ONE} + R_{INT}}$$

Solving for optimum number of turns N gives:

$$N = \frac{\sqrt{Z_P (R_{ONE} + R_{INT})}}{\omega L_{ONE}}$$

Note that RINT was defined as:

$$R_{INT} = M R_{ONE}$$

where M is the coil shape factor.

APPENDIX F: REFERENCES

1. Babani, B.B., ed. 1974. *Coil Design and Construction Manual.* London: Bernards (publishers) Limited.
2. Nelkon, M., & Parker, P. ed. 1970. *Advanced Level Physics*. London: Heinemann Educational Books Ltd.

AN672

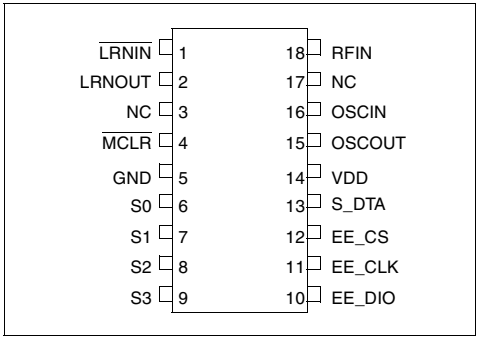
PICmicro® Mid-Range MCU Code Hopping Decoder

Author: Vivien Delport
Microchip Technology Inc.

OVERVIEW

This application note describes the working of a KEELOQ® code hopping decoder implemented on a Microchip Midrange MCUs (PIC16C6X, PIC16C7X, PIC16C62X) The software can be used to implement a stand alone decoder or be integrated with the user application. The decoder supports the Microchip's HCS200, HCS201, HCS300, HCS301, HCS360, and HCS361 KEELOQ hopping code encoders. The decoder supports normal and secure learning. Two manufacturers codes allow different manufacturers to share a public key, but retain their own private keys.

FIGURE 1: MICROCHIP MIDRANGE DECODER



KEY FEATURES

- Supports two manufacturer's codes
- Compatible with Microchip's HCS200, HCS201, HCS300, HCS301, HCS360 and HCS361 encoders.
- PIC16C6X, PIC16C7X and PIC16C62X platforms
- Automatic baud rate detection
- Automatic Normal or Secure learn detection
- Four function outputs
- Six learnable transmitters
- RC Oscillator
- Serial interface

FUNCTIONAL INPUTS AND OUTPUTS

TABLE 1: MICROCHIP DECODER FUNCTIONAL INPUTS AND OUTPUTS

Mnemonic	Pin Number	Input / Output	Function
RF IN	18	I	Demodulated PWM signal from RF receiver. The decoder uses this input to receive encoder transmissions.
LEARN INIT	I	1	Input to initiate learning.
LEARN INDICATION	2	O	Output to show the status of the learn process (in an integrated system this will be combined with the system status indicator).
S_DTA	13	O	Serial data string output which contains the function code, VLOW bit and function code match bit.
S0, S1, S2, S3	6, 7, 8, 9	O	Function outputs, correspond to encoder input pins.
EE_CS EE_CLK EE_DIO	10,11,12	I/O	External Serial EEPROM Interface lines.

KEELOQ is a registered trademark of Microchip Technology, Inc.
PICmicro is a trademark of Microchip Technology, Inc.
Microchip's Secure Data Products are covered by some or all of the following patents:
Code hopping encoder patents issued in Europe, U.S.A., and R.S.A. — U.S.A.: 5,517,187; Europe: 0459781; R.S.A.: ZA93/4726
Secure learning patents issued in the U.S.A. and R.S.A. — U.S.A.: 5,686,904; R.S.A.: 95/5429

PUBLIC AND PRIVATE MANUFACTURER'S CODE

The decoder supports two manufacturer's codes, called the public manufacturer's code and private manufacturer's code. This feature allows two manufacturers to share one public manufacturer's code, but retain their own private manufacturer's code. The decoder uses the public manufacturer's code first to drive the encoder's decryption key, but if learn fails, it will retry using the private manufacturer's code.

PROGRAM FLOW

The decoder software will run on any PIC6C6X/7X with 1K program memory. The operating frequency is 4 MHz. The clock speed is important as the reception routine (RECEIVE) has some critical timing specifications. Other decoder functions that rely on a 4 MHz clock speed are the hold times of the various outputs, time-outs, etc. The compiler used is MPASM.

A high-level description of the main program flow, the transmission validation flow, and the transmitter learn flow are described in the following sections. More detailed descriptions of the other modules can be found in Application Note AN642.

MAIN PROGRAM FLOW

After reset, the decoder enters the main loop where it spends most of the time. The main loop checks the learn button and if pressed (TST_LEARN) enters the learn mode. The microcontroller checks transmissions from the encoders (RECEIVE). Once 65 bits are received, the microcontroller validates the transmission. When a valid transmission is received from a learnt encoder, the decoder sends out a serial data string containing the function code (TX_FUNC) and sets the appropriate function outputs (M_BUT).

TRANSMISSION VALIDATION FLOW

After reception of a code, the decoder will first check if the transmitter is learnt on the decoder. This is done by calculating the checksum on the received transmission's serial number and then searching through the transmitter blocks stored in EEPROM to find a match. If a match is found, the decoder reads the decryption key stored for that transmitter and decrypts the hopping code portion. The 10 LSBs of the discrimination value are compared to the 10 LSBs of the serial number. The 16-bit synchronization counter is validated by checking if the received counter is in the blocked window. The decoder then checks if the counter is in the double operation window. If this is the case, the decoder will wait for the next sequential transmission to synchronize. If the counter is within the single operation window, the decoder updates the EEPROM counters and then generates the appropriate function output.

TRANSMITTER LEARN FLOW

To be able to use a transmitter with the decoder, the transmitter must first be learned into the decoder. Adding a transmitter is done by pressing the learn button. If the button is pressed for longer than 10 seconds, the decoder executes an "erase all" function, which will remove all the transmitters learned.

Normal Secure Learn Selection: In learn mode, the decoder monitors transmissions for 4 seconds. If two codes are received with different serial numbers, the first code is used as the hop code and the second as the seed for the secure learn algorithm. If the two serial numbers are the same, the first received code will be used for the normal learn algorithm.

Manufacturer's Code Selection: The decryption key is derived using the public manufacturer's code stored in a ROM table. The received hopping code is validated by comparing the received transmission's discrimination bits with the lower bits of the serial number. If the decryption validation fails, the decoder will derive the decryption key again by using the private manufacturer's code, also store in a ROM table. If the decryption validation was successful, the decoder will calculate a checksum value on the transmission's serial number. All of the information is then stored in an unused block in the EEPROM. The same memory block will be used if the transmitter was already learned. The result of the learn sequence is displayed on the LED.

COMPILER OPTIONS

Delayed Increment: When this option is enabled, the decoder will automatically increment the synchronization counter by twelve, 30 seconds after the last valid reception. The synchronization window is increased from 16 to 256 in this mode. Delayed increment is used to defeat jamming code grabbers in single button transmitters. This option is enabled by setting the define variable DLY_INC to 1 and recompiling the code.

SERIAL FUNCTION STRING OUTPUT

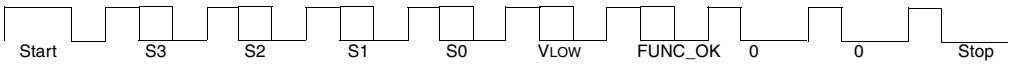
The decoder's serial output sends out a function byte which consists of the function code, battery low status flag, and a function code match bit. After the last bit was clocked out, the line will go high for 500 ms. Repeated transmissions will, as with the binary function outputs, extend this 500 ms time-out. Start bit is one and the stop bit is zero.

The data byte format for this output is shown in Figure 2.

SOURCE CODE

A floppy disk containing the source code for this application note is available under no fee license from your Microchip distributor. The disk order number is DS40149.

FIGURE 2: DATA BYTE FORMAT



Note: The serial data is clocked out with basic pulse width TE = 200 µs

FIGURE 3: MAIN PROGRAM FLOW

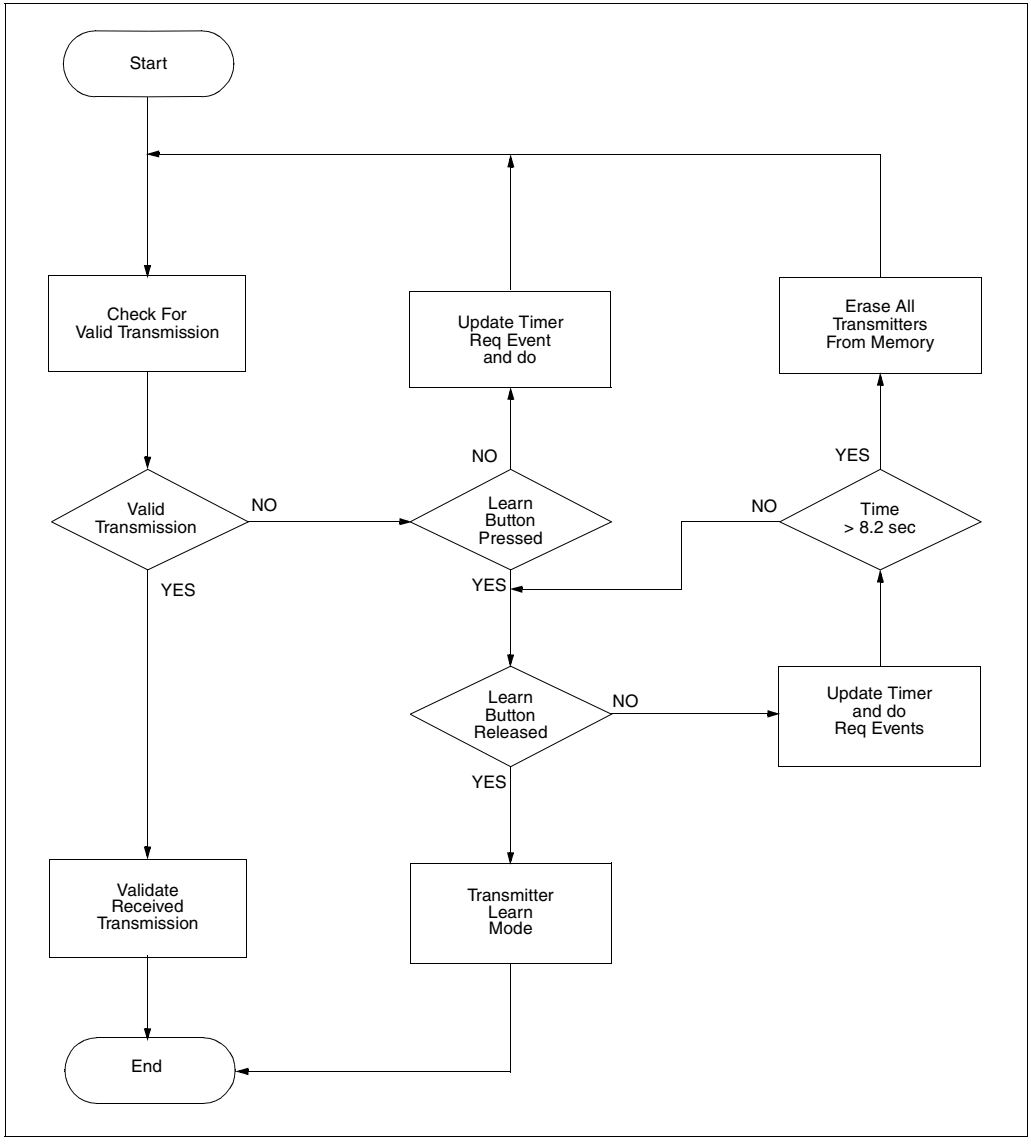


FIGURE 4: TRANSMITTER LEARN FLOW

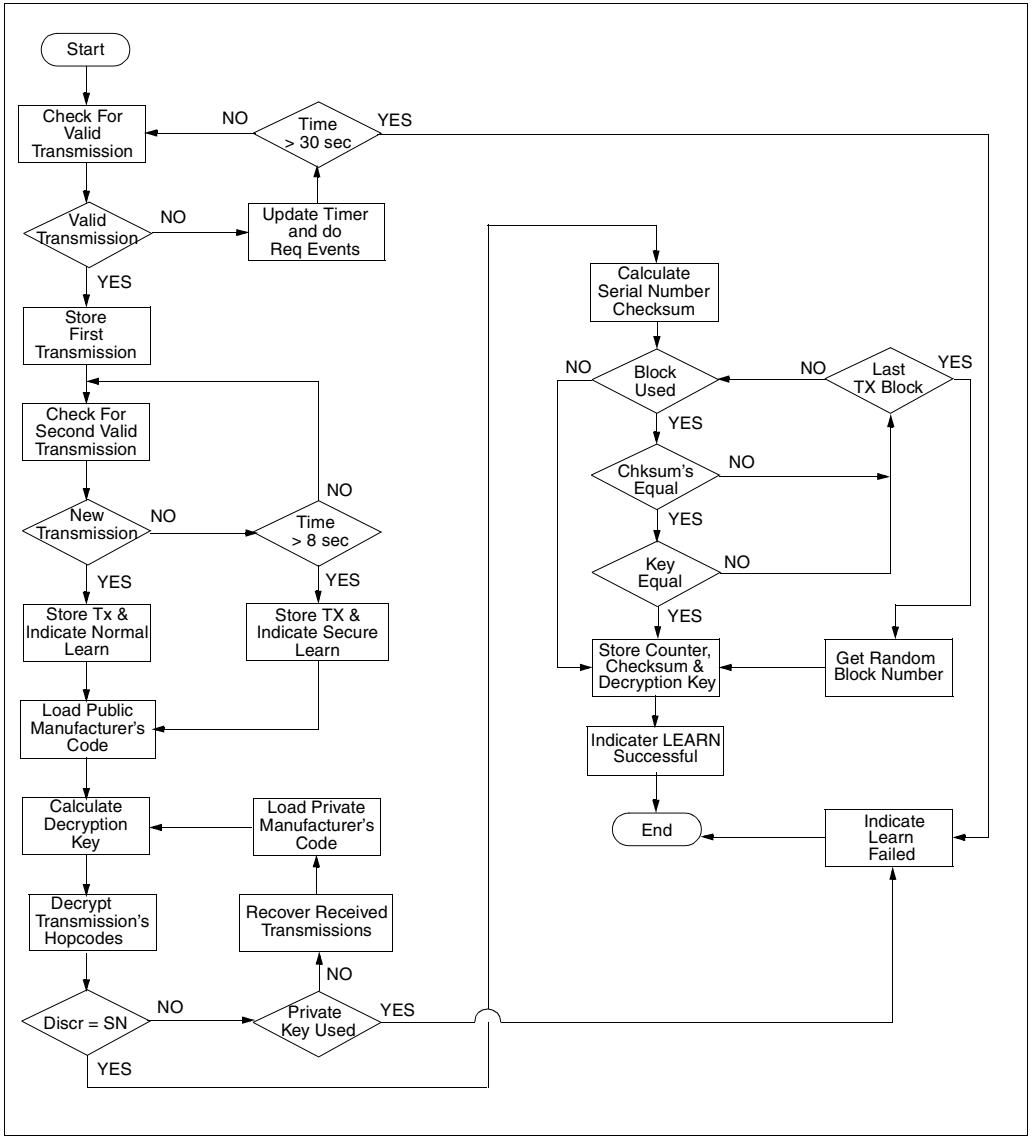
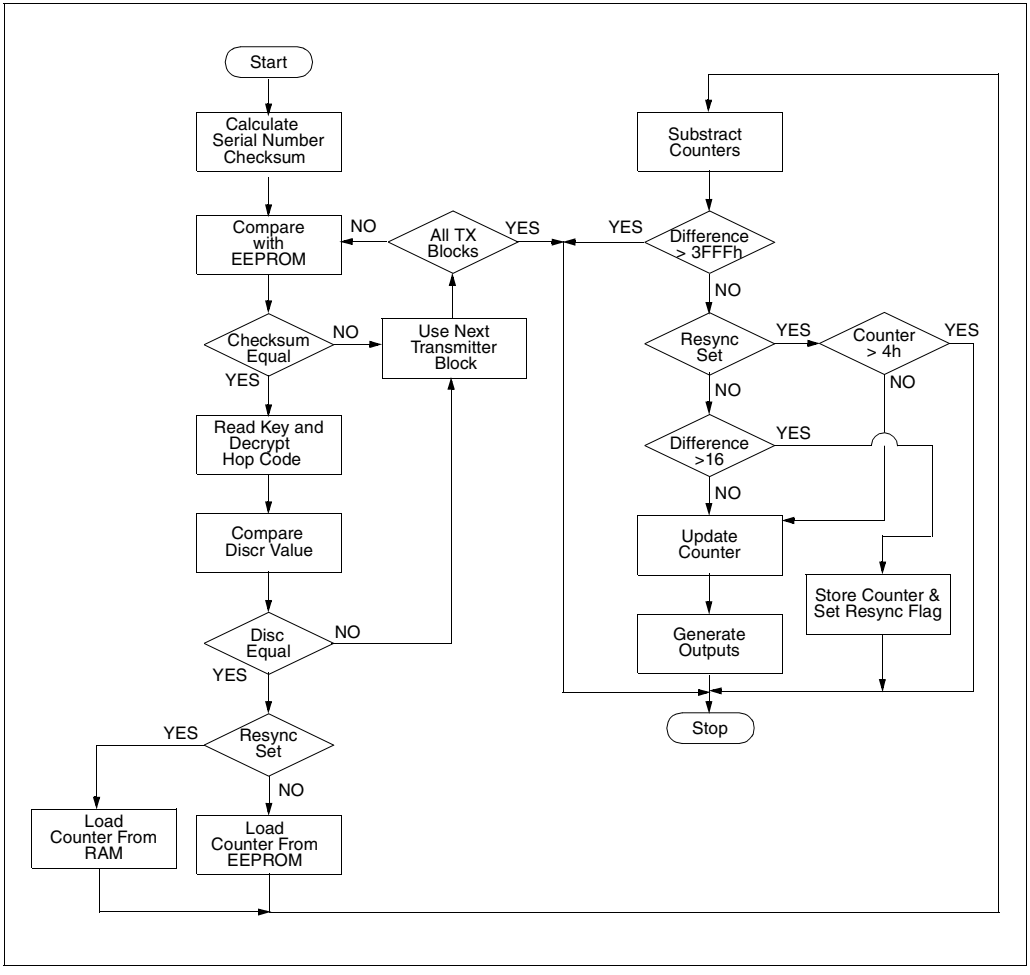


FIGURE 5: TRANSMISSION VALIDATION FLOW



DECODER MEMORY MAPS

TABLE 2: MEMORY MAP ROM (8-BIT BYTES)

Word Address	Mnemonic	Description
42	MKEY1_0	64-Bit Public Manufacturers Code (Used to generate decryption keys)
43	MKEY1_1	
44	MKEY1_2	
45	MKEY1_3	
46	MKEY1_4	
47	MKEY1_5	
48	MKEY1_6	
49	MKEY1_7	
4A	MKEY2_0	64-Bit Private Manufacturers Code (Used to generate decryption keys)
4B	MKEY2_1	
4C	MKEY2_2	
4D	MKEY2_3	
4E	MKEY2_4	
4F	MKEY2_5	
50	MKEY2_6	
51	MKEY2_7	
52	EKEY_0	64-Bit EEPROM Key (Used to encrypt EEPROM data)
53	EKEY_1	
54	EKEY_2	
55	EKEY_3	
56	EKEY_4	
57	EKEY_5	
58	EKEY_6	
59	EKEY_7	

TABLE 3: MEMORY MAP EEPROM (16 BIT WORDS)

Address	Mnemonic	Address	Mnemonic
00	Scratch Pad #1 – First TX	20	CNT20
01	Scratch Pad #1 – First TX	21	CNT21
02	Scratch Pad #1 – First TX	22	DISC2
03	Scratch Pad #1 – First TX	23	CHKSUM2
04	Scratch Pad #2 – Seed TX	24	KEY20
05	Scratch Pad #2 – Seed TX	25	KEY21
06	Scratch Pad #2 – Seed TX	26	KEY22
07	Scratch Pad #2 – Seed TX	27	KEY23
08	Not Used	28	CNT30
09	Not Used	29	CNT31
0A	Not Used	2A	DISC3
0B	Not Used	2B	CHKSUM3
0C	Not Used	2C	KEY30
0D	Not Used	2D	KEY31
0E	Not Used	2E	KEY32
0F	Not Used	2F	KEY33
10	CNT00	30	CNT40
11	CNT01	33	CNT41
12	DISC0	32	DISC4
13	CHKSUM0	33	CHKSUM4
14	KEY00	34	KEY40
15	KEY01	35	KEY41
16	KEY02	36	KEY42
17	KEY03	37	KEY43
18	CNT10	38	CNT50
19	CNT11	39	CNT51
1A	DISC1	3A	DISC5
1B	CHKSUM1	3B	CHKSUM5
1C	KEY10	3C	KEY50
1D	KEY11	3D	KEY51
1E	KEY12	3E	KEY52
1F	KEY13	3F	KEY53

Note:

SCRATCHPAD: Temporary storage of transmission during learn.

CHKSUM: The encoder serial number checksum.

KEY: These bytes contain the decryption key for each encoder.

DIS: Discrimination values and function code storage.

CNT: Two copies of the synchronization counter are stored for each encoder to prevent loss of synchronization information due to EEPROM write failure.

TABLE 4: RAM MEMORY MAP (8-BIT BYTES)

Address	Mnemonic	Description
0C	FLAGS	Decoder flags
0D	ADDRESS	Address register – points to address in EEPROM
0E	TXNUM	Current transmitter's block index
0F	TX_CNT	Transmitter loop counter
10	OUTBYT	General data register, mask register used in decryption
11	CNT0	Loop counters
12	CNT2	
13	CNT3	
14	CNT_HI	16-bit event clock counter
15	CNT_LO	
16	RAM_HI	16-bit RAM counter (used in resynchronization)
17	RAM_LW	
18	TMP0	Temporary registers
19	TMP1	
1A	TMP2	
1B	TMP3	
1C	TMP4	
1D	TMP5	
1E	TMP6	
1F	TMP7	
20	CSR4	64-bit shift register Used in reception, decryption and key generation
21	CSR5	
22	CSR6	
23	CSR7	
24	CSR0	
25	CSR1	
26	CSR2	
27	CSR3	
28	KEY7	64-bit shift register holds decryption key
29	KEY6	
2A	KEY5	
2B	KEY4	
2C	KEY3	
2D	KEY2	
2E	KEY1	
2F	KEY0	

Many of the memory locations in RAM are used by multiple routines. A list of alternate names and functions are given in the table below.

Address	Mnemonic	Also Known As	Description
10	CNT2	OUTBYT	Temporary Loop Counter.
18	HOP1	CSR0	32-bit hop code register.
19	HOP2	CSR1	
1A	HOP3	CSR2	
1B	HOP4	CSR3	
OD	EHOP3	ADDRESS	Extended 32-bit buffer used during key generation as a 32-bit buffer.
1C	EHOP2	TXNUM	
1D	EHOP1	TX_CNT	
1E	EHOP0	CNT3	
17	SER_0	CSR7	28-bit serial number, stores received transmission open 32 bits.
16	SER_1	CSR6	
15	SER_2	CSR5	
14	SER_3	CSR4	
1B	FUNC	CSR3	Button code and user nibble of discrimination value.
1A	DISCR	CSR2	Discrimination value.
19	CNTR_HI	CSR1	16-bit received counter.
18	CNTR_LW	CSR0	

DEVICE PINOUTS

The device used in the application note is a PIC16C6X PDIP.

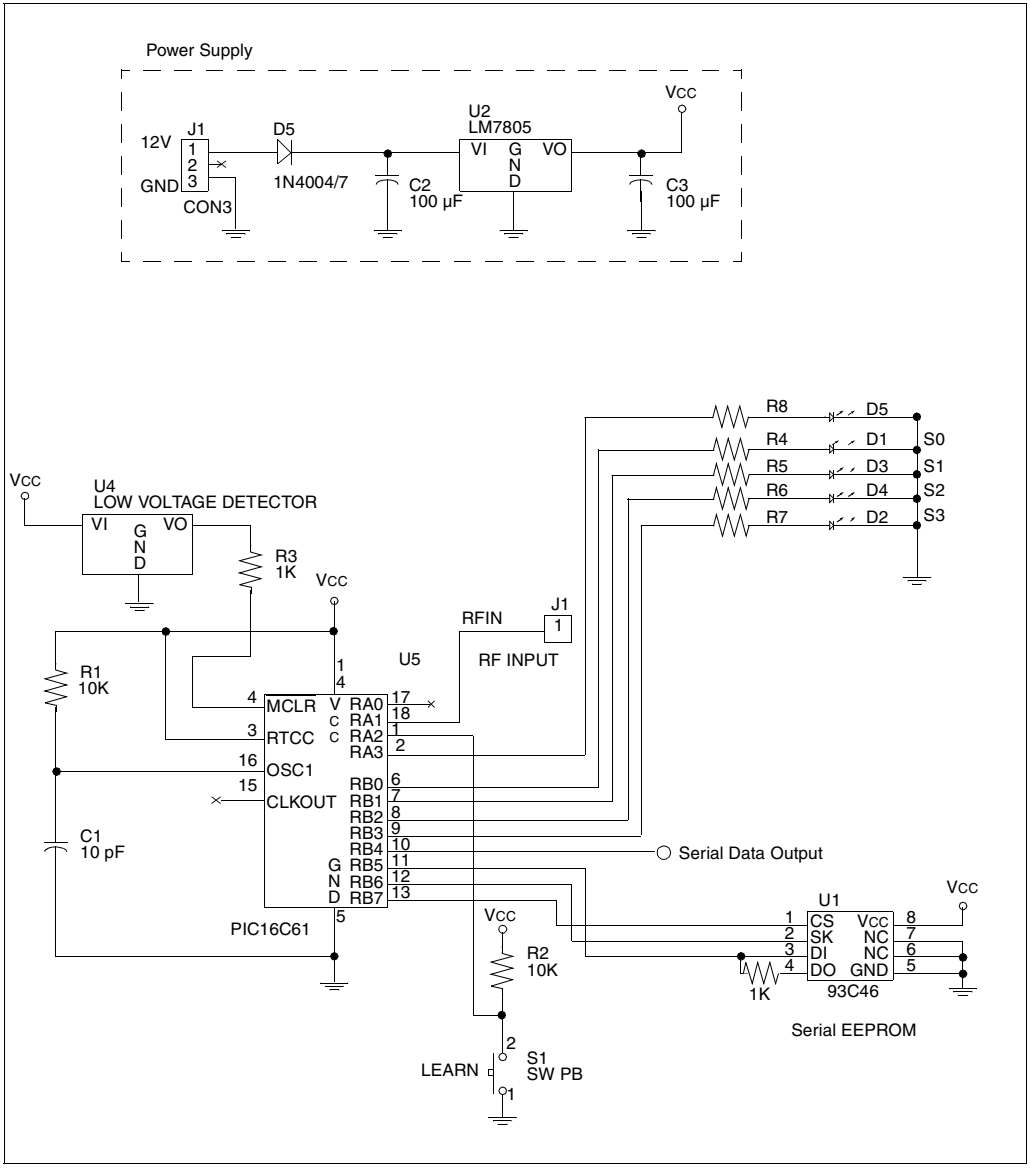
PIN	PICmicro Function	Decoder Function	PIN	PICmicro Function	Decoder Function
1	Port A Bit 2	LEARN Input Act Low	18	Port A Bit 1	RF Input
2	Port A Bit 3	LRN IND Output High	17	Port A Bit 0	Not Used
3	TIMER0	Connect to VDD	16	OSCIN	RC OSC (4 MHz)
4	$\overline{\text{MCLR}}$	Brown-out detect	15	OSCOUT	—
5	GND	Ground	14	VDD	+5V supply
6	Port B Bit 0	S0	13	Port B Bit 7	FUNC OK
7	Port B Bit 1	S1	12	Port B Bit 6	CS (1)
8	Port B Bit 2	S2	11	Port B Bit 5	CLK (2)
9	Port B Bit 3	S3	10	Port B Bit 4	DIO (3+4)

TIMING PARAMETERS

Parameter	Typical	Unit
Output activation duration	524	ms
Output pause if new function code received	100	ms
Erase all duration	8.4	s
Learn mode time-out	33.6	s
Learn successful LED flash duration	4.2	s
Learn successful LED flash rate	3.8	Hz
Learn failure LED on duration	1	s

APPENDIX A: APPENDIX SCHEMATIC DIAGRAMS

FIGURE A-1: SCHEMATIC DIAGRAM OF MICROCHIP KEELOQ DECODER



NOTES:

AN675

KEELOQ HCS410 Transponder Decoder Using a PIC16C56

Author: *Vivien Delport, Mike Sonnabend*
Microchip Technology Inc.

INTRODUCTION

This document describes a secure transponder system. The system is suitable for use in security applications such as cars, motor bikes, and scooters (two-wheelers). Microchip's secure HCS410 KEELOQ® code hopping transponder is used. The decoder is implemented on a Microchip PIC16C56 microcontroller. The software can be used to implement a stand-alone decoder or can be integrated into a security system. The maximum operating range of this particular application circuit is 25 millimeters (one inch).

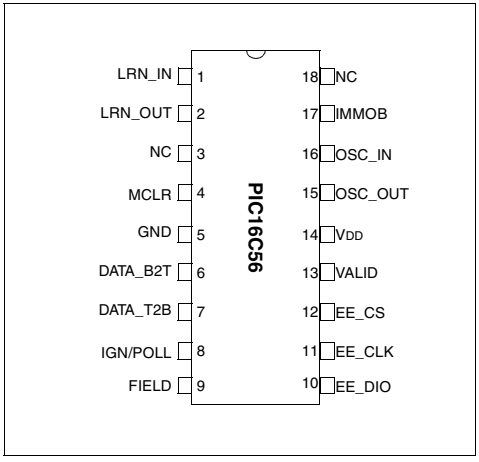
KEY FEATURES

- Stand-alone transponder decoder
- Compatible with KEELOQ HCS410 transponder
- Twelve learnable transponders
- Two function outputs
- XT oscillator

TYPICAL APPLICATIONS

- Automotive/scooter/motorcycle
- Access control
- Gate and garage door openers
- Identity tokens

PIN FUNCTIONS



3

Secure Data Product

KEELOQ is a registered trademark of Microchip Technology, Inc.
Microchip's Secure Data Products are covered by some or all of the following patents:
Code hopping encoder patents issued in Europe, U.S.A., and R.S.A. — U.S.A.: 5,517,187; Europe: 0459781; R.S.A.: ZA93/4726
Secure learning patents issued in the U.S.A. and R.S.A. — U.S.A.: 5,686,904; R.S.A.: 95/5429

HARDWARE

Overview

The hardware for this application note consists of a microcontroller circuit, a transponder, and a base station circuit. Figure 1 shows an overview of the hardware and the interface between each block. The base station is shown in Figure 2. The transponder and microcontroller are shown in Figure 3.

FIGURE 1: TRANSPONDER SYSTEM BLOCK DIAGRAM

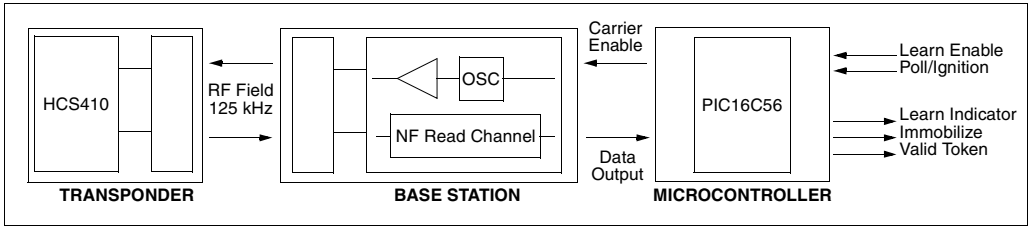


FIGURE 2: READ/WRITE BASE STATION

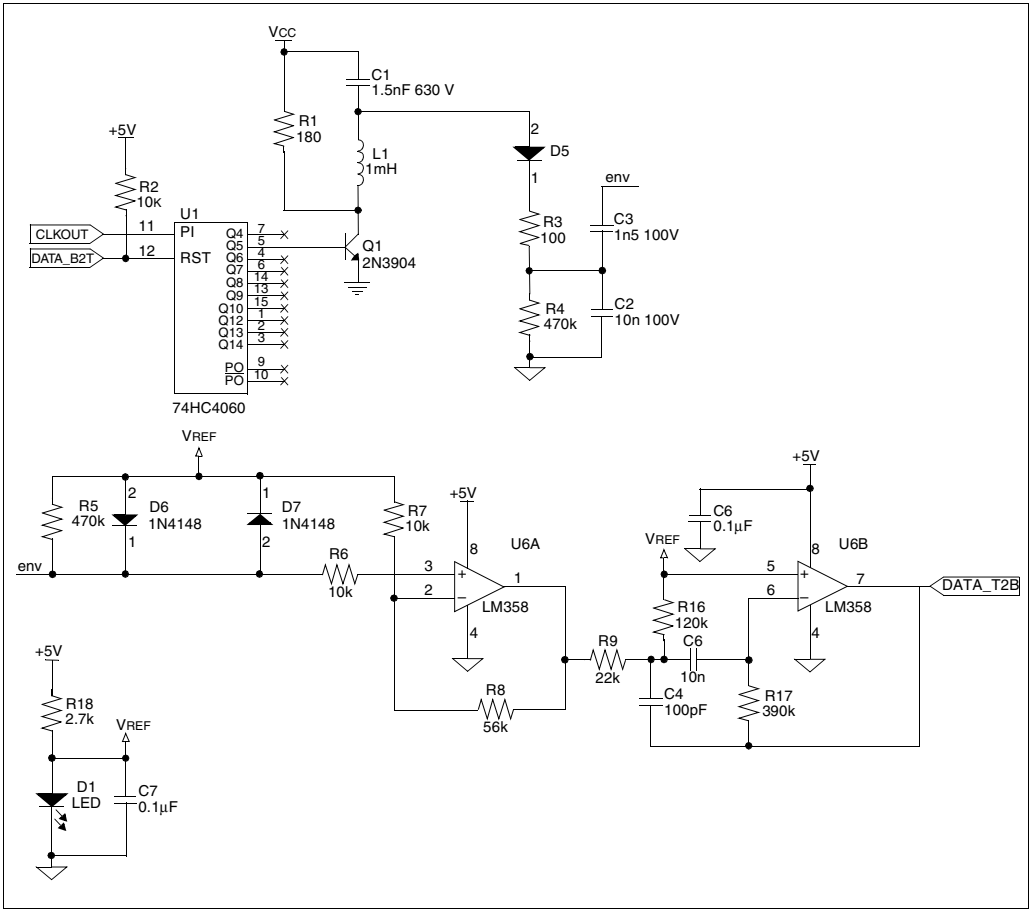
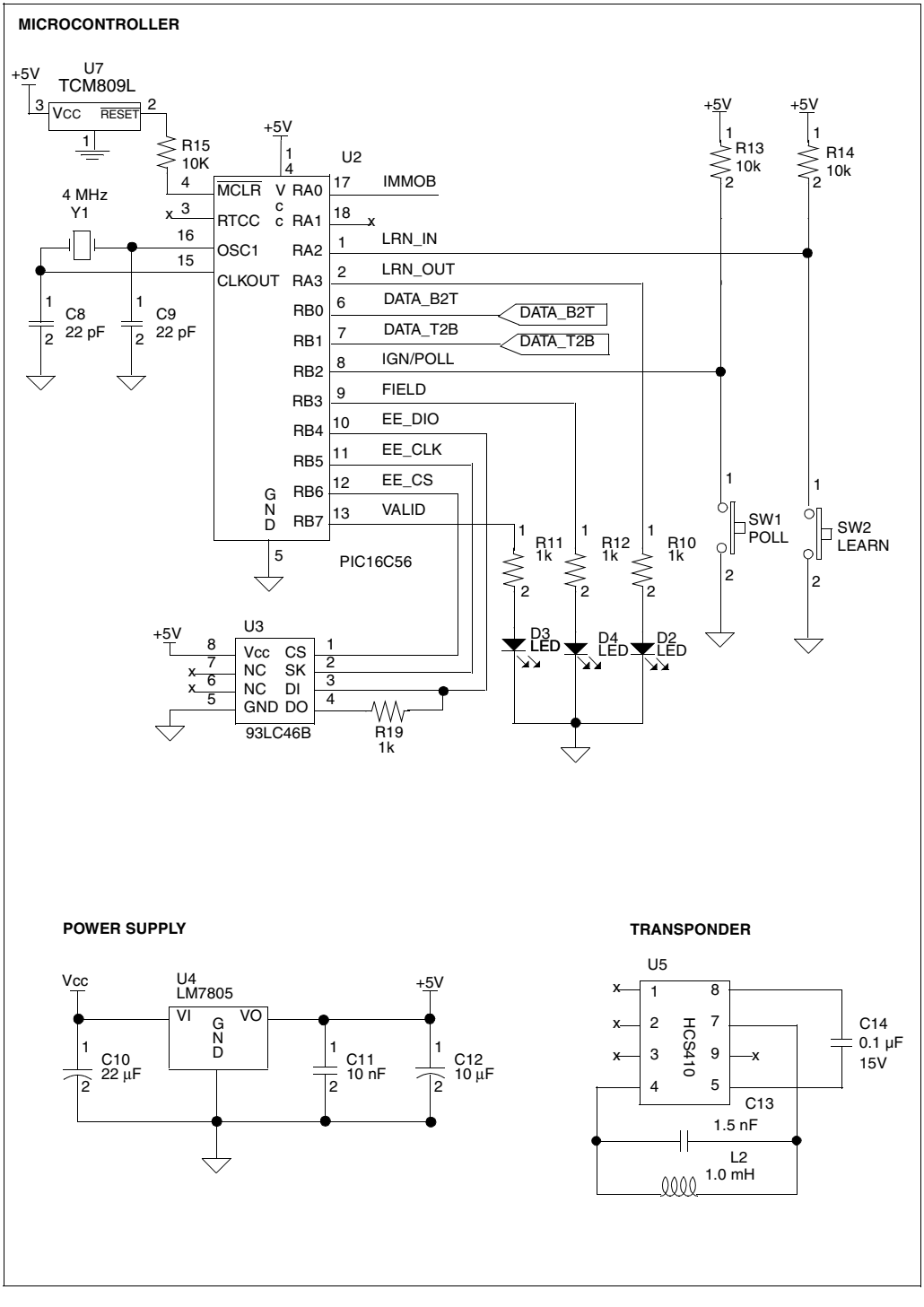


FIGURE 3: MICROCONTROLLER, POWER SUPPLY, AND TRANSPONDER



Microcontroller

The microcontroller consists of the following components:

- Microchip PIC16C56 microcontroller
- A Microchip 93LC46B serial EEPROM used to store all the information of learned transponders
- A 5V supply voltage regulator
- A supply supervisor that inhibits the microcontroller during low voltage events
- Two push buttons used for user inputs
- Three indicator LEDs used for user feedback and indication of function outputs

The microcontroller interfaces to the base station circuit by means of two wires: DATA_T2B used to read data from the transponder to the base station. The carrier enable line of the read/write base station (DATA_B2T) used by the base microcontroller to send data to the transponder.

Table 1 lists the I/O pin assignment for this application.

Read/Write Base Station

The base station is designed to operate from supply VCC between 9V and 12 V.

The 14-bit binary counter U1 divides the 4MHz microcontroller clock CLKOUT to produce a 125kHz clock to transistor Q1. This transistor drives the resonant circuit formed by R1, C1 and L1 to produce a magnetic field. The microcontroller can switch the magnetic field on and off via signal DATA_B2T for communication to the transponder.

The transponder is powered by coupling with the magnetic field produced by L1. The transponder also modulates this field for communication back to the microcontroller. The field modulation is detected at the junction of C1 and L1 by the envelope detector circuit input at diode D5. The envelope detector output signal is amplified by U6A and the data is recovered by bandpass filter U6B. The filter output DATA_T2B is fed directly to the microcontroller.

Table 8 lists the components used in the decoder circuit as shown in the schematics, in Figure 2 and Figure 3.

Table 9 lists the components used for the transponder in Figure 3.

TABLE 1: PIC16C56 I/O PIN ASSIGNMENT

Pin Number	Pin Function	Device Pin	Description
17	IMMOB	RA0	Immobilize function output
1	LRN_ IN	RA2	Input to initiate learning
2	LRN_OUT	RA3	Output to show the status of the learn process (in an integrated system this will be combined with the system status indicator).
6	DATA_B2T	RB0	Data from base station to transponder
7	DATA_T2B	RB1	Data from transponder to base station
8	IGN/POLL	RB2	Input to activate transponder polling
9	FIELD	RB3	Magnetic field is active
10 11 12	EE_DIO EE_CLK EE_CS	RB4 RB5 RB6	Interface lines to external serial EEPROM
13	VALID	RB7	Valid token pulse function output

INTRODUCTION TO THE HCS410 KEELOQ TRANSPONDER

The HCS410 is a KEELOQ code hopping transmitter/ transponder designed for secure entry and identification system. The device combines the circuitry required for Remote Keyless Entry (RKE) and inductively coupled Identify Friend or Foe (IFF) This section describes software which uses the inductive coupled IFF functions of the HCS410.

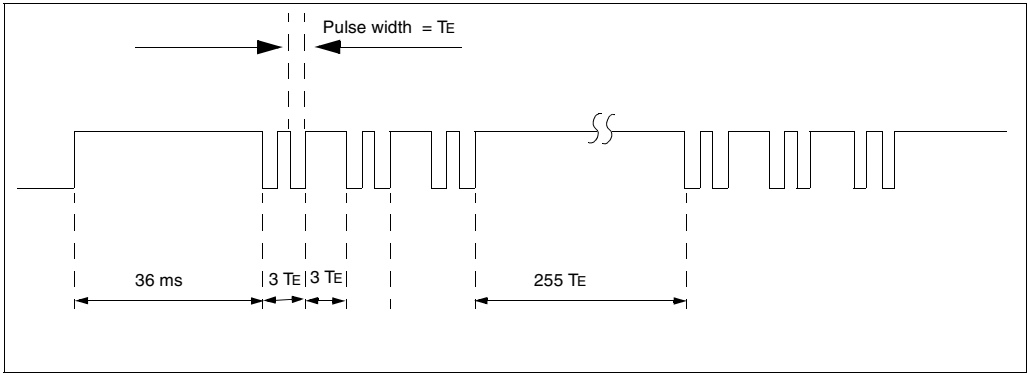
IFF Activation

IFF mode is activated when the HCS410 senses a signal on its LC0 pin. After the HCS410 verifies application of power and elapse of the normal debounce time, the device starts to acknowledge IFF activation by loading the LC pins with continuous acknowledge pulses as shown in Figure 4. This is an indication that the HCS410 is ready to receive a command. All the communication timing is done in multiples of the basic time element TE.

IFF Commands

The HCS410 transponder responds to 5-bit IFF commands or opcodes. The opcodes are sent to the HCS410 with the least significant bit (LSb) first. Depending on the command, additional data may be required for the HCS410 to respond. A list of IFF commands can be found in the HCS410 data sheet (DS40158).

FIGURE 4: IFF ACTIVATION WAVEFORM

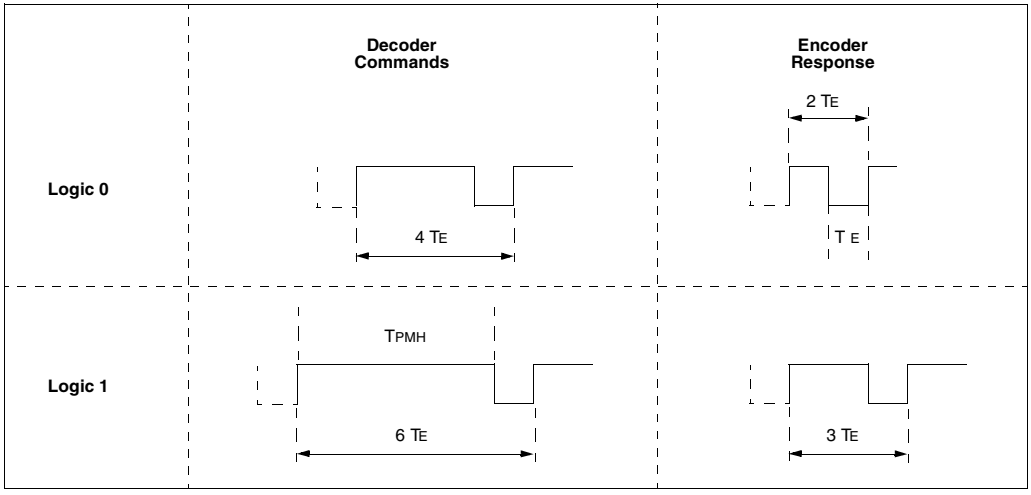


IFF Communication Protocols and Waveforms

All communication to and from the HCS410 during IFF is done in asynchronous Pulse Position Modulation (PPM) format. The format differs when sending commands and data to the HCS410 and when receiving data from the HCS410. After a complete transaction, the HCS410 is ready for the next command and will continue to send out acknowledge pulses. Commands to the HCS410 start with a pulse of 2 TE. Time is measured from rising edge to rising edge with a logic 1 being 6 TE and a logic 0, 4 TE.

Data coming from the HCS410 starts with a start pulse of 1 TE. Again, time is measured from rising edge to rising edge with a logic 1 being 3TE and a logic 0, 2 TE. All data words are preceded by two preamble bits with the logic value 01_2 before the data is sent out.

FIGURE 5: IFF COMMUNICATION WAVEFORM



Identify Friend or Foe (IFF)

Identify Friend or Foe (IFF) is a procedure used to authenticate a transponder. IFF challenges the transponder with a random 32-bit value and then verifies the response.

HCS410 Commands Used

This application uses the following transponder commands: The IFF READ command (Figure 6) is used to read the two portions of the 32-bit serial number (SER1 and SER0).

The IFF CHALLENGE command (IFF1 using key-1 and HOP algorithm) is used to validate the transponder. The microcontroller generates a 32-bit random challenge and then validates the transponder's 32-bit response by decrypting the response using the KEELOQ decryption algorithm.

FIGURE 6: IFF READ COMMAND

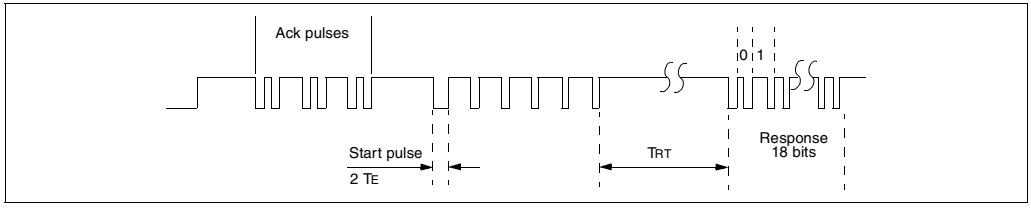
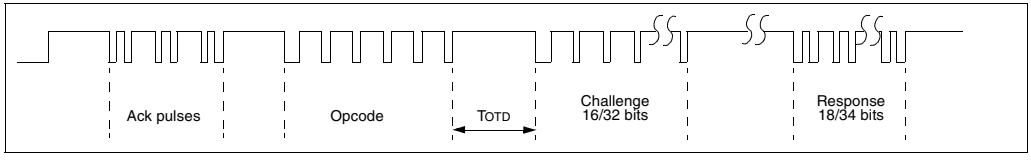


FIGURE 7: IFF CHALLENGE COMMAND



SOFTWARE DESCRIPTION

Overview (Figure 8)

After reset, the decoder enters the main loop. The main loop checks the learn button and if pressed (TST_LEARN) enters the learn mode. The decoder also checks the IGN/POLL input and if low it starts polling for transponder acknowledge pulses for up to 30 seconds. If a transponder is detected, it is validated by means of a 32-bit challenge/response IFF. The decoder pulses the VALID output pin for 500 ms and asserts the IMMOB output for the duration that the IGN/POLL input is held low if the transponder is authentic.

Transponder Validation Flow

The decoder reads the transponder's 32-bit serial number after it detects the acknowledge pulses. It then calculates the 16-bit serial number checksum value. The decoder then searches through all the EEPROM memory blocks for a matching checksum value. Then, it challenges the transponder with a 32-bit random challenge. The decoder validates the transponder by decrypting the 32-bit response with the 64-bit transponder key and comparing it to the 32-bit challenge.

Transponder Learn Flow

The 64-bit Manufacturer's Code is read from the ROM table after the decoder enters learn mode. The decoder then starts polling the field to check if there is any transponder in the field for up to 30 seconds. The decoder reads the transponder's 32-bit serial number after it detects acknowledge pulses. The transponder's decryption key is then calculated using the 64 bit Manufacturer's Code and the 32-bit serial number. The decoder then challenges the transponder with a 32-bit random challenge and validates the 32-bit response by using the newly calculated 64-bit transponder key. The decoder calculates the 16-bit serial number checksum then stores both the 16-bit checksum value and the 64-bit transponder key in EEPROM.

Calibration on Acknowledge Pulses (WAIT_ACK)

The WAIT_ACK function determines if there is a transponder in the field. The routine also calibrates on the acknowledge pulses of the transponder, thereby determining the basic elemental periods T_E, which is used for communication to the HCS410 transponder. The routine switches on the inductive field and waits for 30 ms for the transponder to activate. It then waits for up to 100 ms for a falling edge on the data output line of the read/write base station. The decoder calibrates on the time between the two rising edges. This time, which is equal to 2 T_E, is the used by the WAIT_TE routine during communication to the HCS410. The decoder waits for three acknowledge pulse pairs before it indicates that there is a transponder in the field by setting the zero flag and returning E_OK.

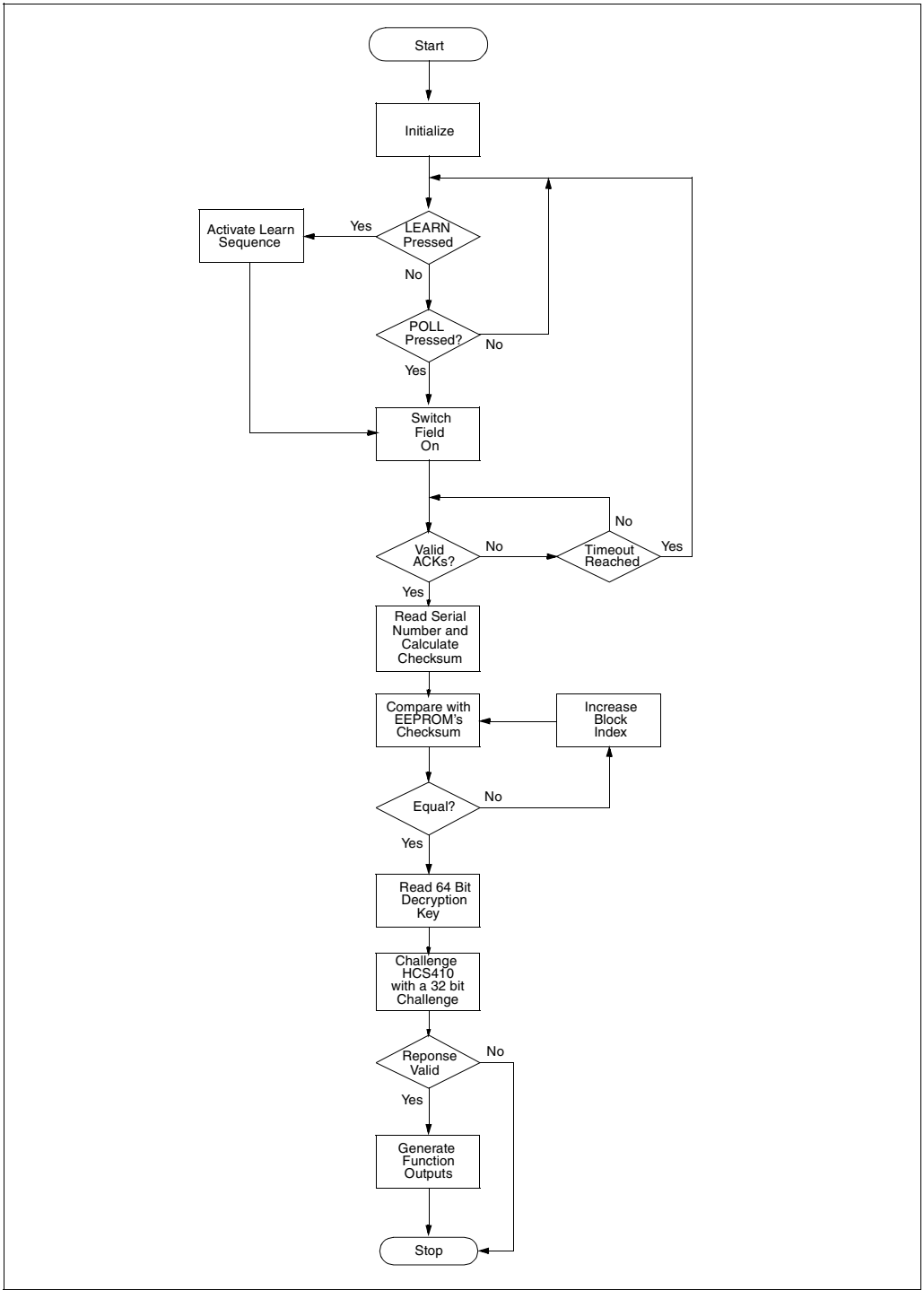
Capturing Data from the HCS410 (REC_PPM)

The REC_PPM function is used to receive PPM data from the transponder. The decoder waits for the start bit, after which it starts measuring the time from rising edge to rising edge. The decoder then checks if this values if less than 2 T_E in which case, the result bit value is set to logic 0. Otherwise, the bit value is set to a logic 1. The function receives either 18- or 34-bits, depending on the initial value of the loop counter (CNT2) The incoming data is stored in a 32-bit temporary shift register (TMP3:TMP0). The two preamble bits are rotated though the shift register and their values are ignored.

Source Code

A floppy disk containing the source code for this application note is available under no fee license from your Microchip distributor. The disk order number is DS40149.

FIGURE 8: PROGRAM FLOW DIAGRAM



ADDING/LEARNING TRANSPONDERS

Overview

Adding/learning a transponder involves calculating the transponder's decryption key, then challenging the transponder and verifying the response using the newly derived key. If the learn was successful, the key and a serial number checksum will be stored in EEPROM. The decoder reads the transponder's 32-bit serial number, forces the upper 4 bits to 6h or 2h to calculate the two input seed algorithms. Then, using these two input seeds and the decryption algorithm, the 64-bit transponder key is calculated. The Manufacturer's Code is stored in a ROM table in program memory.

Generating the 64-bit Key

SEED1 = 6h + 28 bit Serial Number

SEED2 = 2h + 28 bit Serial Number

The transponder key is derived using the KEELOQ decryption algorithm and the 64-bit Manufacturer's Code as follows:

$$Key_{Upper\ 32\ bits} = F_{KEELOQ\ Decrypt}(SEED1) \mid_{64\text{-}Bit\ Manufacturers\ Code}$$

$$Key_{Lower\ 32\ bits} = F_{KEELOQ\ Decrypt}(SEED2) \mid_{64\text{-}Bit\ Manufacturers\ Code}$$

Calculating the 16-bit Checksum From 28-bit Serial Number

The serial number checksum value stored in EEPROM is calculated by as follows:

$$Checksum = [(SER\_3 \oplus SER\_1) << 8] + (SER\_2 \oplus SER\_0)$$

Note: If the calculated checksum is zero the value is changed to 5AA5h.

APPENDIX A: MEMORY ALLOCATIONS

TABLE 2: MEMORY FOR EACH TRANSPONDER

	Name	Description	Bytes
1	Checksum	16-bit checksum value of the serial number	2
2	Key	64-Bit decryption key	8
		Total	**10**

TABLE 3: COMBINED EEPROM MEMORY ALLOCATION

	Name	Description	Bytes
1	Block 1	Scratch pad	6
2	Block 2	16-bit seed counter used by random generator	2
3	Block 3	Stored data for transponder #1	10
4	Block 4	Stored data for transponder #2	10
5	Block 5	Stored data for transponder #3	10
6	Block 6	Stored data for transponder #4	10
7	Block 7	Stored data for transponder #5	10
8	Block 8	Stored data for transponder #6	10
9	Block 9	Stored data for transponder #7	10
10	Block 10	Stored data for transponder #8	10
11	Block 11	Stored data for transponder #9	10
12	Block 12	Stored data for transponder #10	10
13	Block 13	Stored data for transponder #11	10
14	Block 14	Stored data for transponder #12	10
		Total	**128**

TABLE 4: MEMORY MAP EEPROM (16-BIT WORDS)

Address	Mnemonic	Address	Mnemonic
00	CHAL_LW	20	KEY6_2
01	RESP_LW	21	KEY6_3
02	RESP_HI	22	CHKSUM_7
03	CHAL_SEED	23	KEY7_0
04	CHKSUM_1	24	KEY7_1
05	KEY1_0	25	KEY7_2
06	KEY1_1	26	KEY7_3
07	KEY1_2	27	CHKSUM_8
08	KEY1_3	28	KEY8_0
09	CHKSUM_2	29	KEY8_1
0A	KEY2_0	2A	KEY8_2
0B	KEY2_1	2B	KEY8_3
0C	KEY2_2	2C	CHKSUM_9
0D	KEY2_3	2D	KEY9_0
0E	CHKSUM_3	2E	KEY9_1
0F	KEY3_0	2F	KEY9_2
10	KEY3_1	30	KEY9_3
11	KEY3_2	33	CHKSUM_10
12	KEY3_3	32	KEY10_0
13	CHKSUM_4	33	KEY10_1
14	KEY4_0	34	KEY10_2
15	KEY4_1	35	KEY10_3
16	KEY4_2	36	CHKSUM_11
17	KEY4_3	37	KEY11_0
18	CHKSUM_5	38	KEY11_1
19	KEY5_0	39	KEY11_2
1A	KEY5_1	3A	KEY11_3
1B	KEY5_2	3B	CHKSUM_12
1C	KEY5_3	3C	KEY12_0
1D	CHKSUM_6	3D	KEY12_1
1E	KEY6_0	3E	KEY12_2
1F	KEY6_1	3F	KEY12_3

CHAL_LW	Temporary storage of lower 16 bits of challenge
RESP_HI	Temporary storage of upper 16 bits of HCS410's response
RESP_LW	Temporary storage of lower 16 bits of HCS410's response
CHAL_SEED	16 bit seed counter used by random generator to calculate a 32 bit random seed
CHKSUM	Transponder's serial number 16 bit checksum storage
KEY	These bytes contain the 64 bit decryption key for each transponder

TABLE 5: RAM MEMORY MAP (8-BIT BYTES)

Address	Mnemonic	Description
0D	FLAGS	Decoder flags
0E	ADDRESS	Address register – points to address in EEPROM
0F	TXNUM	Current transponder's block index
10	XP_CNT	Transponder loop counter
08	OUTBYT	General data register, mask register used in decryption
09	TMP0	Temporary registers
0A	TMP1	
0B	TMP2	
0C	TMP3	
11	CNT0	General loop counters
12	CNT1	
13	CNT2	
07	CNT3	
14	CSR0	32-bit Code shift register used in decryption and key generation
15	CSR1	
16	CSR2	
17	CSR3	
18	KEY7	64-bit shift register holds decryption key
19	KEY6	
1A	KEY5	
1B	KEY4	
1C	KEY3	
1D	KEY2	
1E	KEY1	
1F	KEY0	

Many of the memory locations in RAM are used by multiple routines. A list of alternate names and functions are given in the table below.

Address	Mnemonic	Also known as	Description
18	DTA1	TMP0	32-bit hop code register
19	DTA2	TMP1	
1A	DTA3	TMP2	
1B	DTA4	TMP3	
OD	EHOP3	ADDRESS	Extended 32-bit buffer used during key generation as a 32-bit buffer
1C	EHOP2	TXNUM	
1D	EHOP1	TE_CNT	
1E	EHOP0	CNT2	
17	SER_0	CSR0	Shift register for unencrypted 32 bits received from transponder
16	SER_1	CSR1	
15	SER_2	CSR2	
14	SER_3	CSR3	

TABLE 6: MANUFACTURER'S CODE IN PROGRAM MEMORY (RETLW TABLE)

Address	Mnemonic	Description
09B	MKEY_0	64-Bit Manufacturer's Code (Used to generate decryption keys)
09C	MKEY_1	
09D	MKEY_2	
09E	MKEY_3	
09F	MKEY_4	
0A0	MKEY_5	
0A1	MKEY_6	
0A2	MKEY_7	

TABLE 7: TIMING PARAMETERS

Parameter	Typical	Unit
Output activation duration	500	ms
Transponder validation duration	330	ms
Erase all duration	4.2	s
Learn mode time-out	25	s

TABLE 8: BILL OF MATERIALS FOR BASE STATION

Item	Reference	Supplier	Part Number	Description
1	C1	Digi-Key	P3499	1.5nF, 630V Polypropylene Capacitor
2	C2	Digi-Key	P4797	10nF, B Series 100V Polyester Capacitor
3	C3	Digi-Key	P4787	1.5nF, B Series 100V Polyester Capacitor
4	C4	Digi-Key	P4773	100pF, B Series 100V Polyester Capacitor
5	C5	Digi-Key	P4797	10nF, B Series 100V Polyester Capacitor
6	C6, C7, C11, C14	Digi-Key	1210PHCT	0.1µF, 50V Axial Ceramic Capacitor
7	C8, C9	Digi-Key	P4016A	22pF, 50V Ceramic Disc Capacitor
8	C10	Digi-Key	P918	22µF 25V, KG Series Miniature Aluminum Electrolytic Capacitor
9	C12	Digi-Key	P6629	10µF 25V, Z Series Miniature Aluminum Electrolytic Capacitor
10	D1, D2, D3, D4	Digi-Key	P403	3mm Red Diffused High Brightness LED
11	D5, D6, D7	Digi-Key	IN4148DICT	100V, 500 MW Fast Switching Diode
12	L1	Coilcraft	DO5022P-105	1mH, DO5022 Series Surface Mount Power Inductors
13	Q1	Digi-Key	2N3904	Small Signal General Purpose Transistor
14	R1	Digi-Key	180R W-1	180R, 5% Metal Oxide Film Resistor
15	R2, R6, R7, R13, R14, R15	Digi-Key	10K W-1	10k, 5% Metal Oxide Film Resistor
16	R3	Digi-Key	100R W-1	100R, 5% Metal Oxide Film Resistor
17	R4, R5	Digi-Key	470K W-1	470k, 5% Metal Oxide Film Resistor
18	R8	Digi-Key	56K W-1	56k, 5% Metal Oxide Film Resistor
19	R9	Digi-Key	22K W-1	22k, 5% Metal Oxide Film Resistor
20	R10, R11, R12, R19	Digi-Key	1K W-1	1k, 5% Metal Oxide Film Resistor
21	R16	Digi-Key	120K W-1	120k, 5% Metal Oxide Film Resistor
22	R17	Digi-Key	390K W-1	390k, 5% Metal Oxide Film Resistor
23	R18	Digi-Key	2.7K W-1	2.7k, 5% Metal Oxide Film Resistor
24	SW1, SW2	Digi-Key	P8006S	Momentary Push-button Switch
25	U1	Digi-Key	MM74HC4060N	14 Stage Binary counter
26	U2	Digi-Key	PIC16C56-XP/P	8-Bit CMOS Microcontroller
27	U3	Digi-Key	93LC46B-I/P	2K CMOS Serial EEPROM
28	U4	Digi-Key	LM78L05ACH	+5V 100 mA Positive Regulator, TO-39
29	U6	Digi-Key	LM358N	Low Power Dual OP Amp
30	U7	Digi-Key	158-2021-2	IC 4.63V UP Reset Monitor SOT-23
31	Y1	Digi-Key	X911	4 MHz ZTA Series Ceramic Resonator

Note: Different value of the same order may have to be used to compensate for tolerance variations in R1, L1, and C1 to keep the peak-to-peak voltage across L1 at 100V.

TABLE 9: BILL OF MATERIALS FOR TRANSPONDER

Item	Reference	Supplier	Part Number	Description
1	U5	Microchip	HCS410	KEELOQ Transponder IC
2	L2	Digi-Key	DN7437	1000 µH Power Axial Inductor
3	C13	Digi-Key	P4787	0.0015 µF 100V Poly B Series CAP
4	C14	Digi-Key	1210PHCT	0.1µF, 50V Axial Ceramic Capacitor

NOTES:

AN677

Designing a Base Station Coil for the HCS410

Author: Mike Sonnabend, Jan van Niekerk
Microchip Technology Inc.

OVERVIEW

This application note describes the Excel spreadsheet to design base station coils. The spreadsheet file name is basesta.xls. A zip file containing this spreadsheet and a copy of this application note can be downloaded from Microchip's web site at www.microchip.com.

The basic approach used is to choose the driver circuit driving voltage and current. These two values are used to calculate the total resistance that the series resistor-inductor-capacitor (RLC) circuit should have. Secondly, the resonating capacitor rated voltage is chosen. The coil inductance and resonating capacitor value can then be calculated.

For a given coil inductance and coil resistance, choosing the coil average diameter and coil winding aspect ratio determines the coil dimensions, number of turns and wire diameter.

The magnetic field strength can be calculated at any given distance given the coil average diameter, number of turns and coil current.

FEATURES

The spreadsheet is split into three worksheets. The first worksheet concerns the HCS410 Evaluation Kit coil driver circuit. Based on the HCS410 Evaluation Kit power supply and coil driver electrical characteristics, the coil inductance, total coil losses at operating temperature and resonant capacitor can be calculated.

The second worksheet uses the coil inductance and total coil losses from the first worksheet with added inputs such as coil diameter to calculate an optimum coil. Coil dimensions, optimum number of turns and wire diameter is provided.

The third worksheet uses the root mean square (RMS) coil current, number of turns and coil diameter from the first two worksheets to calculate the magnetic field at a given axial distance away from the coil.

INTRODUCTION

Overview of Inductive Communication

Communication between a KEELOQ® transponder and a base station occurs via magnetic coupling between the transponder coil and base station coil.

The base station coil forms part of a series RLC circuit. The base station communicates to the transponder by switching the 125kHz signal to the series RLC circuit on and off. Thus, the base station magnetic field is switched on and off.

The transponder coil is connected in parallel with a resonating capacitor (125kHz) and a KEELOQ HCS410 transponder integrated circuit. When the transponder is brought into the base station magnetic field, it magnetically couples with this field and draws energy from it. This loading effect can be observed as a decrease in voltage across the base station resonating capacitor. The KEELOQ transponder communicates to the base station by "shorting out" its parallel LC circuit. This detunes the transponder and removes the load, which is observed as an increase in voltage across the base station resonating capacitor. The base station capacitor voltage is the input to the base station AM-demodulator circuit. The demodulator extracts the transponder data for further processing by the base station software.

Power Losses

- The dominant system power losses in the HCS410 Evaluation Kit are listed below
  - The power supply filter loss, which reduces the coil driver voltage.
  - The losses due to the field effect transistors (FETs) that supply current to the RLC circuit.
  - The coil resistance losses at DC.
  - The coil losses due to skin effect and proximity losses. These are approximated to be equal to the coil resistance at DC.

Using the worksheet

Color coding

Color	Meaning
Green	User input. The default values correspond to the HCS410 Evaluation Kit. If the HCS410 Evaluation Kit is used for a new coil design, changes are not required.
Red	Output
Gray	System defined

Units

The units in the worksheet have been made SI units. Below is a table with some of the most common conversions that the user may come across.

Conversion from:	Operation
Inches (in) to meters (m)	x .0254
Inches (in) to centimeters (cm)	x 2.54
Inches (in) to millimeters (m)	x 25.4
Centimeters (cm) to meters (m)	x 0.01
Millimeters (mm) to meters (m)	x 0.001
Farads (F) to nanofarads (nF)	x 1e-9
Henry (H) to microhenry (µH)	x 1e-6

WORKSHEET 1: HCS410 EVALUATION KIT BASE STATION COIL DRIVER

HCS410 Evaluation Kit Base Station Driver Design

FIGURE 1: EVALUATION KIT COIL DRIVER CIRCUIT

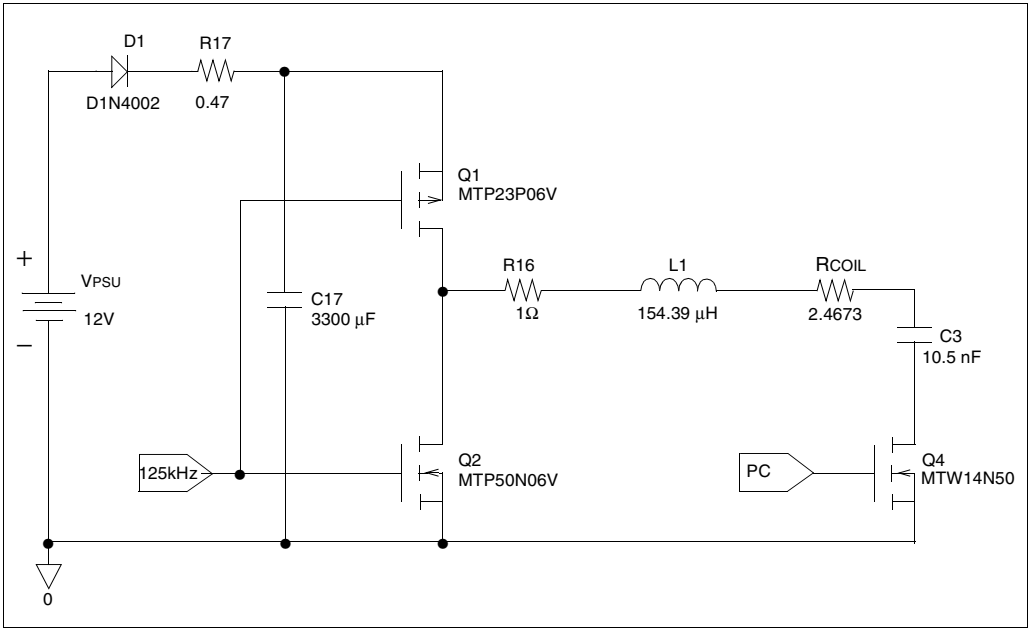


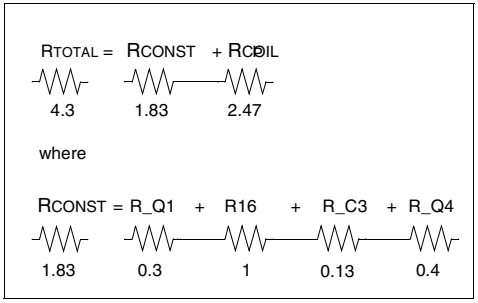
Figure 1 shows the final stage of the evaluation kit coil driver circuit. The input “125 kHz” is a 125 kHz square wave which drives Q1 and Q2 to generate a magnetic field. When this square wave is stopped, no magnetic field is generated. The signal “PC” preserves charge on the capacitor C3 when the field is switched off by disconnecting the capacitor from ground.

The magnetic field produced by a coil is directly proportional to coil current. The base station coil (L1) forms part of a series RLC circuit that resonates at 125kHz. At resonance, the series RLC circuit is a purely resistive load for the driver circuit. Thus the current (and field) is determined as driver voltage divided by RLC circuit resistance. The RLC circuit resistance consists of all the circuit losses and not just the DC resistance of the coil.

The driver square wave peak-to-peak voltage is proportional to the power supply voltage VPSU minus the voltage drop across the blocking diode D1 and filter resistor R17.

The total RLC circuit resistance RTOTAL is fixed by the ratio of driver square wave RMS voltage divided by RMS coil current.

FIGURE 2: RESISTANCE LOSSES



The resistance RTOTAL minus the driver circuit loss RCONST determines the total coil loss resistance RCOIL. The driver circuit loss resistance RCONST consists of the losses due to the FET (Q1 or Q2 and, Q4) "on" resistance, the series resistor R16 if used, and the loss due to the dissipation factor of resonating capacitor C3.

If the starting point for the design selects a power supply current which is too high or a power supply voltage which is too low, then a RTOTAL circuit resistance is required which will be lower than the driver circuit losses RCONST. This is not realizable and would require the coil loss resistance RCOIL to be negative.

For maximum operating distance, the aim of the coil driver is to have low losses. This means using FET's that have a low "on" resistance. Preserving charge in capacitor C3 when the field is switched off reduces the time for the field to build up to its maximum when enabled again. This removes the bandwidth limitation on the Q factor of the resonating circuit, given as Q=f/BW. The Q is now limited by the maximum voltage across the resonating capacitor C3, and is given by Q=VCAP/VRMS, where VRMS is the coil driver voltage applied to the RLC circuit.

Since the Q is limited by the voltage rating of the resonating capacitor V_C3, and the total RLC circuit resistance RTOTAL is known, the coil inductance L is calculated from

$$Q = \frac{V\_C3_{RMS}}{V_{RMS}} = \frac{\omega_r \times L}{R_{TOTAL}}$$

where the resonant frequency f_r is given by

$$f_r = \frac{\omega_r}{2\pi}$$

The coil inductance L and resonant frequency ω_r determine the resonating capacitor C from the equation

$$\omega_r{}^2 = \frac{1}{LC}$$

Data Required

TABLE 1: POWER SUPPLY PARAMETERS

Input	Units	Typical Value	Description
VPSU	[V]	12	Rated PSU voltage used with the base station. This should remain in the range of 8 Volts to 14 Volts if the HCS410 Evaluation Kit Base Station is to be used.
IPSU	[A]	0.5	Rated PSU current. This can be lowered and will lower the magnetic field strength if the design is current limited.

TABLE 2: COIL DRIVER CIRCUIT ELECTRICAL PARAMETERS

Input	Units	Typical Value	Description
f_r	[Hz]	125000	Coil operating frequency (resonance)
V_D1	[V]	0.625	Blocking diode forward voltage drop at IPSU
R17	[ohm]	0.47	Supply filter resistor value
R_Q1	[ohm]	0.3	Maximum "on" resistance of Q1 and Q2
R16	[ohm]	1	RLC series resistor
R_C3	[ohm]	0.128	Resonating capacitor dissipation resistance
R_Q4	[ohm]	0.4	Enhanced frequency circuit Q4 on resistance
V_C3	[V]	400	Resonating capacitor C3 rated maximum voltage

Intermediate Calculations

TABLE 3: COIL DRIVER CIRCUIT CALCULATED PARAMETERS

Output	Units	Typical Value	Description
VDRV	[V]	11.14	Driver square wave peak to peak voltage
VRMS	[V]	5.01	Driver square wave rms voltage
IRMS	[A]	1.11	RMS coil current
RTOTAL	[ohm]	4.51	Total resistance
Q		28.2	Quality factor of RLC circuit
ω_r	[rad/sec]	785398	Transmission frequency
RCONST	[ohm]	1.83	Evaluation circuit losses

Output Data

TABLE 4: RLC RESONATOR CIRCUIT VALUES

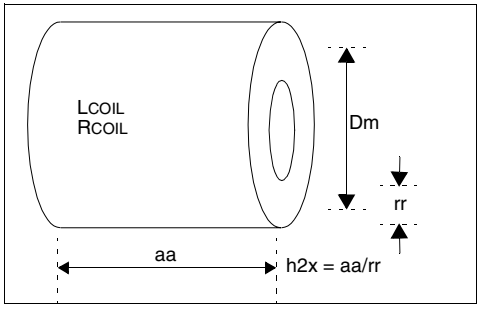
Output	Units	Typical Value	Description
CRES	[nF]	10	Resonating Capacitor
LCOIL	[µH]	162.11	Coil inductance
RCOIL	[ohm]	2.69	Total coil losses at temperature t

The inductance LCOIL and coil resistance RCOIL are used for inputs to the Coil Design worksheet.

WORKSHEET 2: COIL DESIGN ENGINE

Data Required

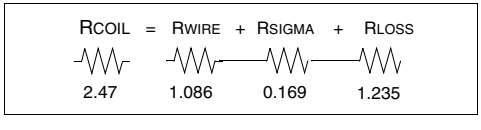
FIGURE 3: COIL DIMENSIONS



The input to the coil design specifies coil inductance LCOIL, coil loss resistance RCOIL, coil average diameter Dm, coil winding aspect ratio h2x, coil loss factor KLOSS, coil wire packing factor, electrical characteristics for the wire used, coil operating temperature and relative permeability for the coil if a core is used.

With large currents, the coil will get hot, as do the drivers. The coil temperature is proportional to the coil losses, which are due to the DC resistance of the wire at the operating temperature t, plus losses due to skin effect and proximity effect. The assumption made in this worksheet is that the losses due to skin effect and proximity effect RLOSS are equal to KLOSS X the DC resistance of the wire at room temperature (RWIRE) plus the increase in wire resistance RSIGMA due to temperature. Thus KLOSS is set to 1 in the worksheet.

FIGURE 4: COIL LOSSES



The default values in Table 5 assume annealed copper wire and an air core coil.

TABLE 5: COIL PARAMETERS

Input	Units	Typical Value	Description
LCOIL	[µH]	162.11	Coil inductance
RCOIL	[ohm]	2.69	Total coil losses at temperature t
Dm	[mm]	54	Coil average diameter
h2x		3	Coil aspect ratio (height/depth)
KLOSS		1	Factor for skin effect and proximity losses. These losses are dissipated in the coil are assumed to be KLOSS times the DC coil resistance at temperature t.
K		0.5	Space factor (packing). This compensates for copper area lost due to wire shape which is round and not square as well as wire insulation. If the coil is wound by hand, then the space factor of less than 0.5 may have to be chosen to compensate for wasted space.
ρ	[ohm-m]	1.72E-08	Coil wire resistivity at 20 degrees C. Resistivity for annealed copper wire is used. If the coil uses another type of wire, then the corresponding resistivity would have to be used.
sigma	[per deg C]	0.00393	Coil wire resistance temperature coefficient. The value used is for copper wire. This value is used to calculate the resistance increase due to the coil operating at a temperature different than 20 ^{o}C.
t	[deg C]	60	Coil operating temperature. This will vary according to the duty cycle, which is determined by the HCS410 Evaluation Kit firmware. The temperature rises with higher duty cycle.
μ_r		1	Relative permeability. It is assumed that the base station coil has an air core. This design does not consider ferrite cores.

Output Data

FIGURE 5: COIL SPECIFICATION

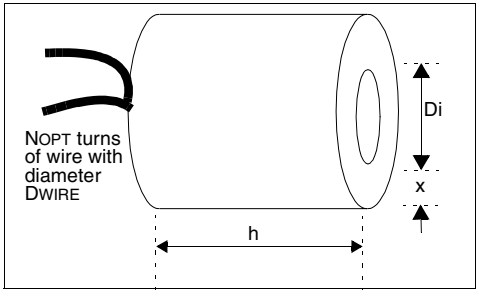


TABLE 6: OUTPUTS

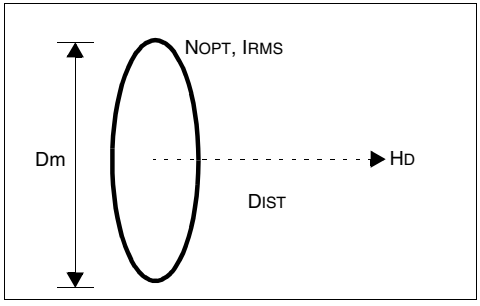
Output	Units	Typical Value	Description
RWIRE	[ohm]	1.16	Coil DC resistance at room temperature
DWIRE	[mm]	0.356	Wire diameter: choose closest to
NOPT	turns	39.59	Optimum number of turns: choose closest to
DI	[mm]	52.38	Coil inside diameter
h	[mm]	4.86	Coil axial height
x	[mm]	1.62	Coil winding depth

WORKSHEET 3: MAGNETIC FIELD PRODUCED BY A COIL

Data Required

For a base station coil shown below

FIGURE 6: MAGNETIC FIELD AT DISTANCE DIST



The magnetic field at distance DIST along the axis is given by

$$HD = \frac{N_{OPT} \times I_{RMS} \times \left(\frac{D_m}{2}\right)^2}{2 \times \left[\left(\frac{D_m}{2}\right)^2 + (\text{Dist})^2\right]^{3/2}}$$

The values NOPT and Dm are used from the coil design on worksheet 2 and IRMS is used from worksheet 1. The input Dist can be entered to see what the magnetic field HD is at a certain distance. The value range is the estimated range at which the field would activate an Evaluation Kit long-range transponder

TABLE 7: TRANSPONDER DISTANCE FROM BASE STATION

Input	Units	Typical Value	Description
DIST	[cm]	0	Transponder axial distance from coil center

Output Data

TABLE 8: MAGNETIC FIELD STRENGTH

Output	Units	Typical Value	Description
HD	[A/m]	814.26	Magnetic field at distance Dist, in ampere turns per meter.
Range	[cm]	24.11	Evaluation Kit transponder, proximity activation range. This is the distance along the coil axis where the field is 1.123 ampere-turns per meter, which is the field, required to activate an Evaluation Kit transponder for RF talkback.

CONCLUSION

By using the formulas given in Appendix B, the equation for the field can be rewritten as shown in the following equation.

$$HD = 5 \times 2^{3/4} \times \sqrt{5} \times \sqrt{127} \times \frac{\sqrt{V_{CAP}}}{\sqrt{\mu_r \times \omega_r}} \times \sqrt{\frac{V_{RMS}}{R_{TOTAL}}} \times \left(D_m \times \sqrt{\frac{3 \times D_m + 9 \times h + 10 \times x}{\left(D_m^2 + 4 \times \text{Dist}^2\right)^3}} \right)$$

It can be seen that the field is:

- proportional to the square root of the rated capacitor voltage VCAP,
- proportional to the square root of the current in the coil and,
- inversely proportional to the cube of the axial distance from the coil.
- The reason that increased frequency ω_r, or increased relative permeability μ_r decreases the field is because the number of turns has to be decreased to remain within VCAP specification.
- For a distance Dist, it can be shown that the magnetic field strength HD is a maximum when Dm (coil radius) is twice the distance Dist.

APPENDIX A: EXAMPLE CALCULATION

Problem

Design a coil that uses the HCS410 Evaluation Kit as base station, has a diameter of 120mm with square coil cross section, and draws 1A from the power supply.

Solution

Worksheet 1: Change the following from the default values in the worksheet.

IPSU=1amp, R16=0 ohms, as a series resistor is not needed.

The coil inductance required is 81 µH with resonating capacitor 20nF.

Worksheet 2: Change the following from the default values in the worksheet.

Dm=120, h2x=1 to get a square coil cross section.

The result for the coil is to use wire with a diameter of 0.48mm. From the table, AWG #24 is chosen which has a diameter of 0.51 mm. The number of turns is 17.

Worksheet 3:

The distance for a standard Evaluation Kit long-range transponder to be activated should be 39cm.

The values calculated give a good starting point for the coil design but are approximations, and the resonating capacitor will still have to be trimmed for resonance to occur. The model used for the losses is KLOSS is equal to 1. This loss factor may vary for different coils.

APPENDIX B: FORMULAS USED

This appendix gives the main formulas used in the worksheet. All values use metric units.

For a square wave with peak to peak voltage VDRV, driving an RLC circuit, the RMS value of this voltage VRMS is given by

$$V_{RMS} = V_{DRV} \times \frac{\sqrt{2}}{\pi}$$

The total resistance of the circuit is given by

$$RTOTAL = \frac{VRMS}{IRMS}$$

For a frequency f in Hertz, the radians per second frequency is given by

$$\omega_r = 2\pi f$$

For a series RLC circuit with resistance RTOTAL, coil with inductance LCOIL and resonating capacitor with rated voltage V_CRES, the quality factor Q of the circuit is given by

$$Q = \frac{V\_C_{RES}}{V_{RMS} \times 2 \times \sqrt{2}} = \frac{\omega_r L_{COIL}}{R_{TOTAL}}$$

The resonating capacitor CRES value is given by

$$C_{res} = \frac{1}{\omega_r^2 \times L_{COIL}}$$

If NOPT turns of wire occupies a cross sectional area of x by h, with packing factor of K (ratio of copper area to total area), then the wire diameter DWIRE is

$$D_{WIRE} = 2 \times \sqrt{\frac{K \times x \times h}{\pi \times N_{OPT}}}$$

For a coil of average diameter DM, wound with NOPT turns of wire with diameter DWIRE and resistivity ρ, the resistance of the wire RWIRE is given by

$$R_{WIRE} = \frac{4 \times \rho \times D_m \times N_{OPT}}{D_{WIRE}^2}$$

For a coil of average diameter DM, with core which causes relative permeability UR, wound with NOPT turns of wire with axial height h and radial depth (inside radius to outside radius) x, the coil inductance in Henry is given by

$$L_{COIL} = \frac{\mu_r \times (D_m)^2}{127000 \times (3 \times D_m + 9 \times h + 10 \times x)} \times (N_{OPT})^2$$

For a coil of average diameter DM, wound with NOPT turns and carrying current IRMS, the magnetic field at axial distance DIST away is given by

$$H = \frac{N_{OPT} \times I_{RMS} \times \left(\frac{D_m}{2}\right)^2}{2\left[\left(\frac{D_m}{2}\right)^2 + (Dist)^2\right]^{3/2}}$$

APPENDIX C: REFERENCES

1. Babani, B.B., ed. 1974. *Coil Design and Construction Manual*. London: Bernards (publishers) Limited.
2. Nelkon, M., & Parker, P. ed. 1970. *Advanced Level Physics*. London: Heinemann Educational Books Ltd.

Note: Our design does not calculate self inter-winding capacitance of the inductor.

COMMENTS

The authors welcome feedback, comments, questions and errata via e-mail.

mike.sonnabend@microchip.com

jan.van.niekerk@microchip.com

GLOSSARY

Dissipation Factor: A measure of the losses of a capacitor. Dissipation factor varies with frequency and temperature.

Proximity Effect Losses: These are losses caused by adjacent conductors (proximity) generating eddy currents in each other.

Relative Permeability µr: The ratio of magnetic field in a material to the magnetic field if the material were replaced by vacuum.

Skin Effect: This is the tendency for an alternating current to flow near the surface (skin) of a conductor as the frequency increases.

NOTES:

AN714

Wireless Home Security Implementing KEELOQ® and the PICmicro® Microcontroller

Author: *Richard L. Fischer*
Microchip Technology Inc.

INTRODUCTION

This application note describes a Microchip system solution for a low end/power wireless home security system. This design implements an HCS200 encoder for the intruder sensor signal encryption, one PIC12C508A PICmicro® for sensor monitoring and RF signal initiation, HCS515 decoders for decrypting the received intruder sensor signal and a PIC16C77 PICmicro for base station panel monitoring and control. Other support logic is included, such as a battery back-up circuit, simple single stage lead acid battery charger and external siren control, but the focus of the application is the implementation of Microchip KEELOQ® and PICmicro products for a complete solution.

APPLICATIONS

Applications implementing low power RF wireless systems are entering the marketplace with ever increasing acceptance, fueled in part by growing awareness of the consumer. Low power wireless systems usually transmit less than 1mW of power and do not require user licenses for operation. These systems operate over distances of 5 to 100 meters, depending on the application.

Wireless systems are being implemented in the automotive, residential, personal and commercial arenas with increasing growth rates every year. Wireless systems in these areas include, but are not limited to: vehicle alarm arming and disarming, home garage and gate door openers, home lighting control, home security and fire alarm systems, pagers, cellular phones, utility meters for near-field readings, warehouse inventory control systems and RF LANs.

In many of these applications, different levels of security are required. The level of security required is dependent on the application and customer demands. For instance, a warehouse inventory control or utility meter system may require little or no security features whereas automobile access and home security alarm systems will require more.

No matter what level of security features are implemented, one vulnerable link in low power RF wireless based security systems is the actual RF signal itself. An RF based system could allow for the would be intruder/thief to use a code scanning or a code grabbing system to possibly gain unauthorized access to the home, car or other less secure system.

Code scanning is an effective tool for the would be thief on systems with limited number of possible code combinations which are found in quite a number of remote control systems. Patience, time and a hand-held microprocessor based system are all the intruder would need.

Code grabbing is a far easier way of gaining unauthorized access. In this method, the thief would monitor and capture the RF signal used in opening the home garage door or car. The thief would then wait until an opportune moment and then retransmit this code to gain access.

It is apparent that secure remote control systems can be implemented, if two conditions are met. The KEELOQ code hopping system meets both these conditions with ease.

1. A large number of possible combinations must be available.

   A 66-bit transmission code is used to make scanning impossible. The 32-bit encrypted portion provides for more than 4 billion code combinations. A complete scan would take 17 years! If the 34-bit fixed portion is taken into account, the time required for a complete scan jumps to 5,600 billion years.

2. The system may never respond twice to the same transmitted code.

   The random code algorithm will never respond to the same code twice over several lifetimes of a typical system.

Every time a remote control button is pushed, the system will transmit a different code. These codes appear random to an outsider, therefore, there is no apparent relationship between any code and the previous or next code.

For more information on code scanning, code grabbing and an introduction to KEELOQ Code Hopping, see Technical Brief TB003, titled “An Introduction to KEELOQ Code Hopping”. Refer to the Secure Data

Products Handbook, Microchip document number DS40168 for additional information on KEELOQ® products.

With the arrival of the Microchip KEELOQ code hopping security products, secure remote control systems can be implemented. Microchip provides a complete security solution with a full range of encoders and decoders that incorporate the Company's patented KEELOQ code hopping technology algorithm, allowing you to get the most advanced security technology. KEELOQ encoders are designed to be the transmitters and KEELOQ decoders, the receiver of secure remote keyless entry (RKE) systems.

The KEELOQ encoders feature on-chip, error corrected EEPROM for non-volatile operation and, therefore, reduce the required components normally external to the encoder. The only additional circuitry required are push buttons, battery and RF circuitry.

The KEELOQ decoders are single-chip solutions that employ normal and secure learning mechanisms, and operate over a wide voltage range. Microchip decoders are also full-featured with serial interface to PICmicro microcontrollers, allowing designers to integrate the decoder with system functionality.

SYSTEM OVERVIEW

The Microchip KEELOQ solution is being implemented into more and more systems requiring proven security. Systems such as, but not limited to:

- Automotive security
- Gate and garage door openers
- Identity tokens
- Software protection
- Commuter access
- Industrial tagging
- Livestock tagging
- Parking access
- Secure communications
- Residential security

One simple example implementing the KEELOQ solution is a home security system. The home security system described herein utilizes KEELOQ code hopping security products and a PICmicro microcontroller.

Some specific system design goals for this low end/power security system were:

- Wireless solution
- Secure RF transmissions
- Battery operation of intruder sensors for a minimum of 1.5 years
- Sensor module flexibility to operate with various off-the-shelf switches for doors and windows
- Microcontroller based system
- Battery back-up system which provided for up to 10 hours of operation at a load draw of 400mA
- System remote arm and disarm by means of the existing garage door opener (not completely implemented in the current release)

The three main hardware components, which comprise this home security system are, the Base Station Panel, Intruder Sensor Modules and the Battery Charger/Accessory Unit.

SYSTEM DESCRIPTION

The following sections provide a greater in depth look into each of the three main hardware components.

BASE STATION PANEL

The home security base station panel provides for:

- Monitoring of sensor module initiated RF signals
- User interface and system setup via the 4x4 keypad
- Visual feedback via the 2x16 character Liquid Crystal Display (LCD) module
- On-board piezo buzzer control
- Real-time clock
- Monitoring of a single stage battery charger unit
- Automatic DC power selection circuit

The base station can be functionally divided into 4 main components:

1. KEELOQ HCS515 decoder interface.
2. Power supply switching circuit.
3. Battery charger unit monitoring.
4. LCD and 4x4 keypad interface.

Base Station Operation

One of the more important tasks the base station's microcontroller (PIC16C77) must handle, is to monitor and process the output data of the two HCS515 decoders. Each decoder is capable of learning up to seven sensor modules or "zones". Within each zone, there are four different message types which the PIC16C77 must decode and process (See Appendix A, Figure 6 for the following text description).

For example, a sensor module may send an alarm, okay, test or learn transmission. In turn, the PIC16C77 reads the data (up to 80-bits) from the HCS515 decoder, evaluates the message contents and initiates the appropriate action. If an alarm condition occurs, the external siren will be activated and the internal panel piezo buzzer, (BZ1) will sound, if enabled. For any valid signal reception, such as a test, learn, sensor okay condition or alarm transmission, the history profile for that sensor module will be updated. This update consists of a time stamp and the sensor's module battery status. If the sensor battery status indicates a low battery state, then the base panel piezo buzzer will beep (if enabled) four times every minute until the condition is resolved. The user can determine which sensor module battery is low through proper keypad selections and individual zone battery status displayed on the LCD.

The base station can be placed into a “learn” mode so as to learn up to seven sensors (zones). Through proper keypad selections, the PICmicro commands the HCS515 decoder into the learn mode. (See Figure 1 and Table 1). Once placed in this mode, two consecutive transmissions by the sensor are required to complete a successful learn. Once a sensor is learned, a “key” name for that zone must be selected. A menu will automatically appear on the LCD for this selection process. Currently up to 15 different key names are available to choose from. The selected key name is then stored in the HCS515 EE user space.

The history profile of each sensor is written to the available user EEPROM in the HCS515 decoder. The total EEPROM data space available in the HCS515 is 2Kbits. System data space is 1Kbits and user memory space is the remaining 1Kbits. System data space is not accessible by the user (See Table 2 for the user EEPROM memory map). The demodulated data input into the decoders is obtained from a super regenerative data receiver referenced RF1 (See Appendix A, Figure 7, Part Number RR3-433.92 - Manufactured by Telecontrolli). The receiver has a typical RF sensitivity of -105dBm and consumes 3mA, maximum.

A Microchip microcontroller supervisory circuit, MCP130-475, is used to ensure the required system panel operating voltage range is adhered to. The brown-out feature on the PIC16C77 was not used since the base panel system operating voltage range is 4.5 to $5.5V_{DC}$.

The base station panel is designed to operate from one of two available DC input sources: the converted AC line power or the 12V lead-acid battery back-up (See Appendix A, Figure 5 for the following text description).

Both DC sources are fed into the panel via connector, JP1. From JP1, each source is input to separate adjustable voltage regulators. The primary DC source regulator, U2, has its Vout set to $5.50V_{DC}$, while the secondary DC source regulator, U3, has its Vout set to $5.05V_{DC}$. Both regulator outputs are fed into separate inputs of the automatic battery back-up switch, U1.

Switch U1, is an 8-pin monolithic CMOS I.C. which senses the DC level of the two input sources and connects the supply of the greater potential to its output, pin 1. This is a break-before-make switch action and switching typically occurs in 50µs. Capacitor C9 is used to minimize the switching transient during the transition.

One limitation of the switch is its current switching capabilities. Maximum continuous current of the switch is exceeded by this panel design so two PNP transistors were added which provides for greater power switching.

The implementation of the PNP transistors is such that when the primary source is the greater of the two, pin 6 of U1, labeled “PBAR”, is effectively tied to ground internally and therefore Q1 is biased into saturation. During this configuration, Q3 is in the off state because pin 3, labeled “SBAR”, is at hi-impedance.

When the secondary DC source is the greater of the two, Q3 will be biased into saturation and Q1 will be off. In either state, the load is handled through the transistors and the “VO” pin of U1 is no longer required. However, the “VO” pin is configured for driving LEDs, which indicate the DC source selected.

The PIC16C77 receives status back relating to the switch selection via the signal labeled “PSOURCE”. The state of this feedback signal is active low when the primary DC source is selected, and active high if the secondary source is selected.

This power switching circuit also allows for the PIC16C77 to select the secondary source, even if the primary source is present. If the signal labeled “BATSEL” is asserted high by the PIC16C77, NPN transistor Q2 will be turned on and effectively reduce Vout of U2 to $1.25V_{DC}$. U1 will detect the drop and switch to the backup source. This feature can be used as a test-mechanism. Finally, V_{OUT} of U3 supplies the voltage reference, V_{REF}, for the Analog-to-Digital module on the PIC16C77. This signal is labeled “VBAT”.

As with any home security system, it is important to provide for backup power in the event of a primary source failure. A simple single stage back-up/charger unit is provided for this requirement. Based upon a load draw of 400mA, 10 hours of operation are provided for. This is a worse case scenario, which includes a 170mA (typical) current draw from the external siren.

The PIC16C77 samples the battery voltage, once per minute. If the sampled battery voltage is less than $\sim 12.75V_{DC}$, then the current limit resistor, R15, is switched in or if $>12.75V_{DC}$, then bypassed (See Appendix A, Figure 9 and Appendix A, Figure 10). The user can view the battery voltage on the LCD by pressing the appropriate keys on the 4x4 keypad. (See Table 1).

The system LCD and 4x4 keypad provide for system status feedback and setup. Status information, such as sensor module battery state, zone faults and time-of-day are displayed on the 2x16 character LCD. The LCD is updated by the PIC16C77 through data transfers on PORTD (See Appendix A, Figure 6 and Appendix A, Figure 7).

System parameter setup such as enabling the internal piezo buzzer, time-of-day setup, zone naming and alarm initiating is provided through the 4x4 keypad. System test modes are also entered through the keypad. The keypad is interfaced to PORTB which utilizes the interrupt on change feature.

FIGURE 1: *4x4 Keypad Layout*

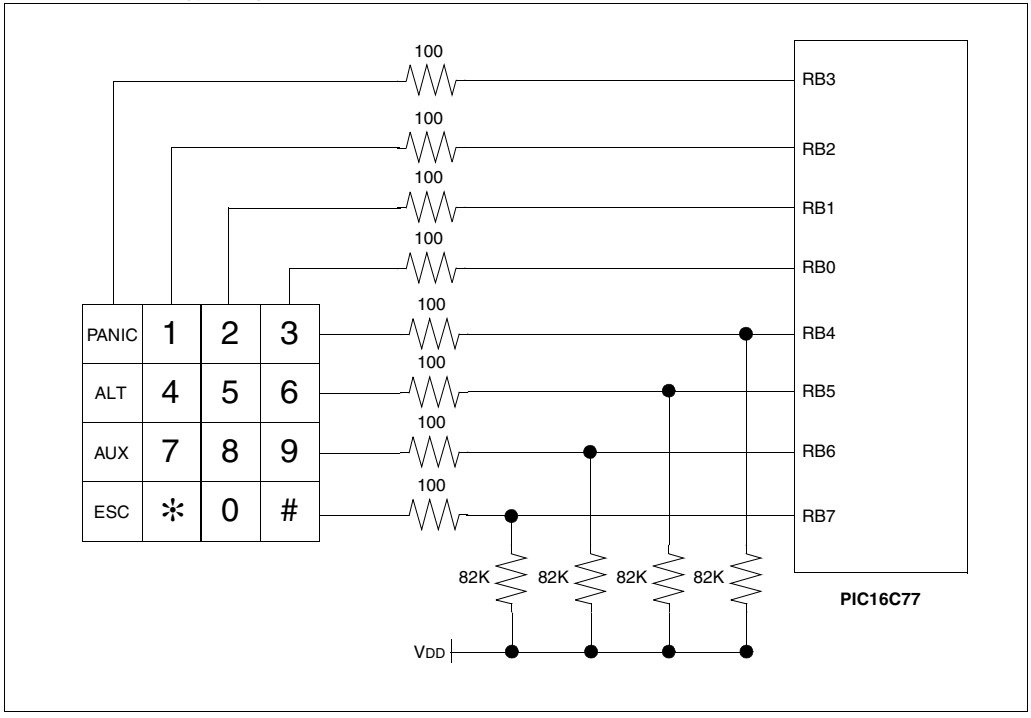


TABLE 1: *4x4 Keypad Selections versus Respective System Response*

Primary 4x4 Keypad Entry	Secondary 4x4 Keypad Entry	Final 4x4 Keypad Entry	System Response
PANIC	*	N/A	Arm System Immediately w/o entry of User Code and w/o Arm time delay
	#	N/A	Arm System via entry of User Code and enable arm time delay (5 minutes)
	1	N/A	Enable Internal Piezo Buzzer to sound if selected
ALT	5	1	Select Battery as Power Source to System
		2	Monitor Battery Voltage if primary is Selected
		3	Review Battery on/off cycle time and daily cycle count
	6	1	Review number of learned transmitters (sensor modules/ zones)
		2	Review sensor module battery status and check time-of-day last received
	7	N/A	Check on Alarm conditions for system. (was an Alarm signal received)
AUX	6	1	Place HCS515 decoder in 'Learn' mode and execute
		2	Place HCS515 decoder in 'Erase All' mode and execute
	7	1	Toggle if time-of-day will be displayed on LCD
		2	Set / Change time-of-day via keypad entries Keys 1 & 4 for incrementing/decrementing hours count Keys 2 & 5 for incrementing/decrementing minutes count Keys 3 & 6 for incrementing/decrementing seconds count
	8	N/A	Entry of 4-digit User Code. The 4-digit Master Code must be known and entered before the User code can be changed. Master code in ROM via SQTP
	9	N/A	Set time for key wait expiration.
ESC	#	N/A	Disable System Armed State with Entry of User Code
	0	N/A	Clear LCD Screen
	1	N/A	Disable Internal Piezo Buzzer from sounding if selected
	PANIC	N/A	Clear Alarm Zone Trip Status for LCD
	ESC	N/A	Toggle LCD Backlight

TABLE 2: *HCS515 Decoder User EEPROM Map*

ADDR	Description	ADDR	Description	ADDR	Description	ADDR	Description	ADDR	Description
80	USER_COD1	9E	ZONE4_NM	BC	XMTR_CNT	DA		F8	ALRM_HRS
81	USER_COD1	9F	TOD4_HRS	BD		DB		F9	ALRM_MIN
82	USER_COD1	A0	TOD4_MIN	BE		DC		FA	ALRM_SEC
83	USER_COD1	A1	TOD4_SEC	BF		DD		FB	ALRM_STAT
84	USER_COD1	A2	BATT4_ST	C0	USER_COD2	DE		FC	
85		A3		C1	USER_COD2	DF		FD	
86	MSTR_CODE	A4		C2	USER_COD2	E0		FE	
87		A5	ZONE5_NM	C3	USER_COD2	E1		FF	LAST_XMIT
88		A6	TOD5_HRS	C4	USER_COD2	E2			
89	ZONE1_NM	A7	TOD5_MIN	C5		E3			
8A	TOD1_HRS	A8	TOD5_SEC	C6		E4			
8B	TOD1_MIN	A9	BATT5_ST	C7		E5			
8C	TOD1_SEC	AA		C8		E6			
8D	BATT1_ST	AB		C9		E7			
8E		AC	ZONE6_NM	CA		E8			
8F		AD	TOD6_HRS	CB		E9			
90	ZONE2_NM	AE	TOD6_MIN	CC		EA			
91	TOD2_HRS	AF	TOD6_SEC	CD		EB			
92	TOD2_MIN	B0	BATT6_ST	CE		EC			
93	TOD2_SEC	B1		CF		ED			
94	BATT2_ST	B2		D0		EE			
95		B3	ZONE7_NM	D1		EF			
96		B4	TOD7_HRS	D2		F0	BT_ON_CNT		
97	ZONE3_NM	B5	TOD7_MIN	D3		F1	BT_ON_HRS		
98	TOD3_HRS	B6	TOD7_SEC	D4		F2	BT_ON_MIN		
99	TOD3_MIN	B7	BATT7_ST	D5		F3	BT_ON_SEC		
9A	TOD3_SEC	B8		D6		F4	BT_OFF_HRS		
9B	BATT3ST	B9		D7		F5	BT_OFF_MIN		
9C		BA		D8		F6	BT_OFF_SEC		
9D		BB		D9		F7			

LEGEND:

USER_CODx	User Code (2 locations)
ZONEx_NM	Zone Name (where x is the zone number)
TODx_HRS	Time of Day (Hours, where x is the zone number)
TODx_MIN	Time of Day (Minutes)
TODx_SEC	Time of Day (Seconds)
BATTx_ST	Battery Status (Sensor Module Battery) - 0xF0 (High) - 0x0F (Low)
BT_ON_CNT	Daily count for battery cycles (on/off)
BT_ON_HRS	Time of Day (hours) when battery is last selected
BT_ON_MIN	Time of Day (minutes) when battery is last selected
BT_ON_SEC	Time of Day (seconds) when battery is last selected
BT_OFF_HRS	Time of Day (hours) when battery is last de-selected

LEGEND: (Continued)

BT_OFF_MIN	Time of Day (minutes) when battery is last de-selected
BT_OFF_SEC	Time of Day (seconds) when battery is last de-selected
ALRM_STAT	Alarm Status - 0x41 (Alarm) - 0xBE (Clear)
ALRM_HRS	Alarm Condition (hours) when alarm is activated
ALRM_MIN	Alarm Condition (minutes) when alarm is activated
ALRM_SEC	Alarm Condition (seconds) when alarm is activated
XMTR_CNT	Number of transmitters learned
LAST_XMIT	Last decoder transmission type received and recorded
MSTR_CODE	Currently not used

A 32.768KHz watch crystal is connected to the Timer1 external oscillator pins for the generation of a real time clock. Specific system data, such as alarm time, battery on/off cycle time and all valid decoded RF signals are time-tagged. The clock time is setup/changed via the 4x4 keypad and operates using the military time format, i.e., 2:30PM will display as 14:30:00, while 2:30AM will display as 02:30:00.

INTRUDER SENSOR MODULE

The four main functions of the intruder sensor modules are:

1. Intruder detection.
2. Sensor battery status.
3. Sensor learn.
4. Sensor test.

For each of these functions, an input to the HCS200 KEELOQ encoder is asserted (active high) by the PIC12C508A. The end result is a 66-bit encrypted code word transmission, via RF, to the base station panel for decryption and processing.

In order to provide for these functions, additional logic is implemented to complement the HCS200 encoder. The logic consists of a PIC12C508A microcontroller, a relaxation type oscillator circuit, N-channel MOSFET for signal level translation, Colpitts oscillator used for the Amplitude Shift Keying (ASK) transmitter, and a few additional passive components.

See Appendix A, Figure 11 and Figure 12 for the following sensor operation discussion.

One important operational requirement of the sensor module besides reliable signal decoding and secure RF transmission, is low current consumption. With the components selected, sensor battery life is calculated to be a minimum of 1.5 years.

Sensor operation

The KEELOQ HCS200 encoder is a perfect fit for implementation into the sensor modules. The HCS200 encoder provides for:

Security

- Programmable 28-bit serial number
- Programmable 64-bit encryption key
- Each transmission is unique
- 66-bit transmission code length
- 32-bit hopping code
- 28-bit serial number, 4-bit function code, VLOW indicator transmitted
- Encryption keys are read protected

Operating

- 3.5–13.0V operation
- Three button inputs
- Seven functions available
- Selectable baud rate
- Automatic code word completion
- Battery low signal transmitted to receiver
- Non-volatile synchronization data

Other

- Easy to use programming interface
- On-chip EEPROM
- On-chip oscillator and timing components
- Button inputs have internal pulldown resistors

The HCS200 combines a 32-bit hopping code generated by a powerful non-linear encryption algorithm, with a 28-bit serial number and 6 information bits to create a 66-bit transmission stream. The length of the transmission eliminates the threat of code scanning and the code hopping mechanism makes each transmission unique, thus rendering code capture-and-resend schemes useless. (See Figure 2, Figure 3 and Figure 4 for code word organization and formats).

The encryption key, serial number and configuration data are stored in EEPROM, which is not accessible via any external connection. This makes the HCS200 a very secure unit.

FIGURE 2: *Code Word Organization.*

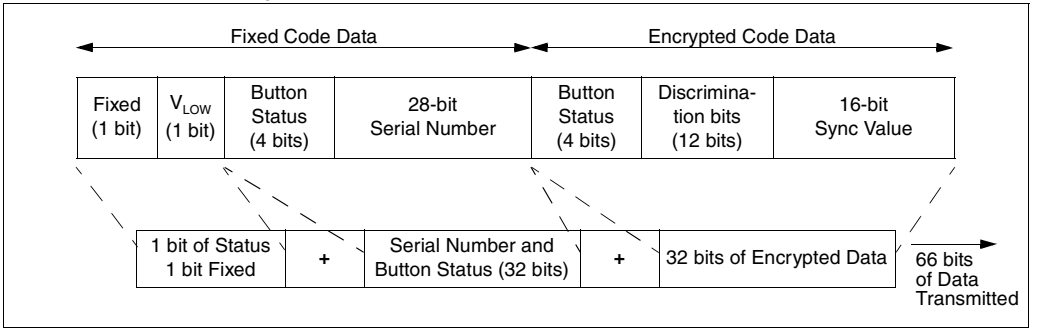
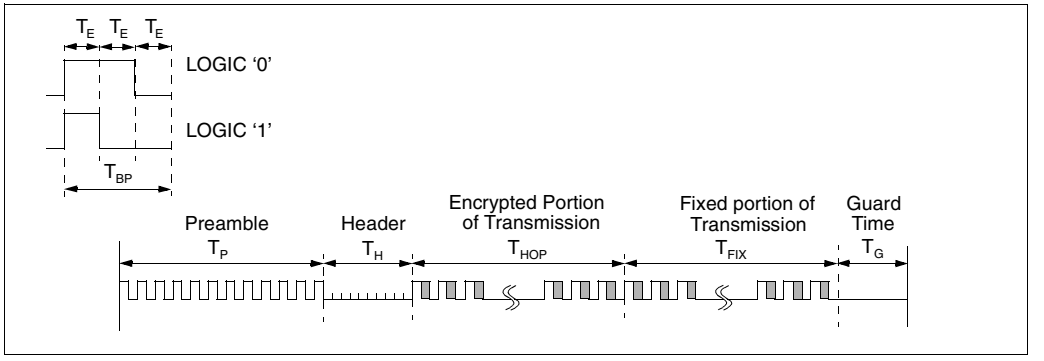


FIGURE 3: *Code Word/PWM Transmission Format .*



T_E - Basic pulse element *

T_{BP} - PWM bit pulse width *

T_P - Preamble duration *

T_H - Header Duration *

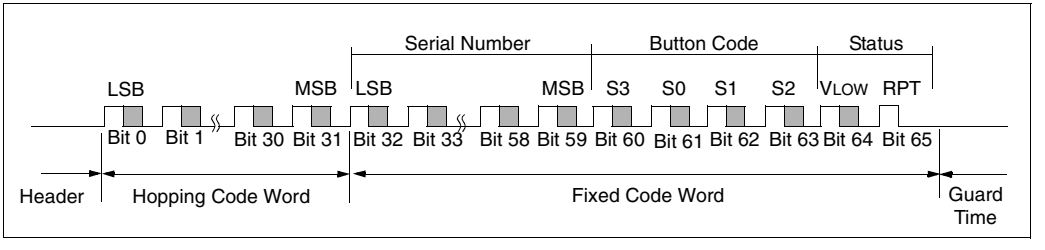
T_{HOP} - Hopping code duration *

T_{FIX} - Fixed code duration *

T_G - Guard Time *

* - See Data Sheet (DS40168) for parameter timing specifics

FIGURE 4: *Data Word Format.*



The HCS200 responds to input signals initiated by the PIC12C508A. The PIC12C508A provides for the sensor signal detection and decoding and RF signal initiation. The PIC12C508A is configured to operate on the internal RC oscillator with the wake-up on pin change feature enabled. The PIC12C508A is placed in the "sleep" mode for about 99% of the time, based on the overall repeated time period of 1.5 hrs (discussed later). While the wake-up on pin change feature is utilized, the internal weak pull-ups are disabled and larger ohmic external resistors are used. This reduces the current consumption, while retaining the wake-up on pin change feature.

Since the HCS200 and RF circuitry are only required after the PIC12C508A awakens by a pin state change and with the requirement to reduce additional current draw from the battery, the HCS200 and RF circuitry are powered through I/O pin, GP5. The current sourcing capability of the PIC12C508A is sufficient for this requirement. This configuration reduces the overall current draw by 1μA (typically) during sleep mode.

The PIC12C508A detects and responds to one of the four input pin state changes, which are:

1. Intruder sensor activation on input pin GP3 (active high).
2. Sensor test transmission activated by switch closure on input pin GP1 (active high).
3. Sensor learn transmission activation by switch closure on input pin GP1 (active high).
4. 1.5 hr timing cycle on input pin GP0 (active high). This signal is used to generate a sensor battery status transmission.

Once the wake-up signal has been decoded, the HCS200 and RF circuitry are powered-up via pin GP5, labeled "CNTPWR". (See Appendix A, Figure 11 and Figure 12). A 3μs delay is allowed for power-up stabilization and then the PIC12C508A asserts an active high signal to U2 inputs S0, S1 or both, depending on the wake-up signal decoded. The HCS200 input pin states are as follows:

1. Pin S0 asserted only - alarm condition.
2. Pin S1 asserted only - 1.5 hr elapsed time sensor update.
3. Pin S0 and S1 asserted simultaneously, Learn or test mode entered.

The alarm condition is in response to a possible intruder detection at the door or window. The switches used for monitoring door and window access are FORM C and SPST type, respectively. The FORM C door switches used are specifically designed for steel skin doors, but are well suited for use in wooden doors. SENTROL, INC manufactures both switch types used. The door and window switch part numbers used are 1078C and 3650W, respectively.

Jumpers JP1 and JP2 are configured, based on whether the sensor is to be used for a door or window. If the sensor is used for a door, JP1 is closed and JP2 is open. For a window application, JP2 is closed and JP1 is open. These jumpers can be used for implementing different resistor values, based upon the sensor switch implemented.

It is imperative that the correct switches are specified to eliminate a source of false alarm conditions. Items such as door to frame gap and door material construction contribute a big part in selecting the appropriate switch sensor.

The 1.5 hr elapsed time sensor update is developed using a relaxation timing circuit. The timing circuit consists of a JFET configured as a constant current source set to 400nA, a Programmable Unijunction Transistor (PUT), an N-channel MOSFET for signal level translation and a reverse biased diode to reduce PUT discharge time.

This timing circuit is configured to produce a state change on pin GP0 approximately every 100 seconds. The constant current source charges up the low leakage 10μF capacitor, C1. When the voltage across C1 equals the firing voltage of the PUT, which is the peak point emitter voltage termed V_P, and if the current is large enough, the PUT will enter into the negative resistance region and begin to discharge. The maximum firing current required by the 2N6028 for a RG value = 1M is 150nA. RG = (R2*R3)/(R2+R3).

Resistors R2 and R3 set the voltage V_P. This voltage is V_P = ~(VBat * R2)/(R2 + R3). Diode D1, which is reversed biased during the PIC12C508A sleep period is used to reduce the PUT discharging time period. When the PIC12C508A wakes from sleep, diode D1 is forward biased and provides a low impedance path to ground for C1 discharge (See Appendix A, Figure 11). When the diode is not used, the discharge period was observed to be about 7-8ms. With the diode, the discharge time period was reduced to tens of microseconds. The savings of several milliseconds, reduces the time the PIC12C508A is awake and therefore helps to extend the battery life of the sensor module.

The N-channel MOSFET, Q2, provides for signal level translation from the PUT. If the voltage level set by resistors R2 and R3 (~ 4VDC for new batteries) is applied directly to pin GP0, additional current consumption would be realized, since this voltage on a Complimentary Metal Oxide Semiconductor (CMOS) input would be near its threshold.

The N-channel MOSFET is configured as a switch, such that the drain channel is tied to VBat through a 6.8Mohm resistor and the source channel is grounded. Then, by tying the drain channel to pin GP0, the voltage on GP0 is either VBat or ground, depending on the PUT state applied to the gate of Q2.

Changing the R2 to R3 ratio could increase voltage, V_P. If the voltage level was set such that it falls outside the CMOS input threshold, then Q2 and R1 could be eliminated.

When the PIC12C508A wakes from sleep, it increments a counter variable and then returns to sleep. This process repeats until the counter variable equals 54, which equates to approximately 1.5 hrs. At this 1.5 hr time cycle, the PIC12C508A initiates an 'OKAY' signal. This signal is received and decoded by the base panel for determining the state of the sensor module battery.

The PIC12C508A then resets the cycle count variable to zero and starts the time cycle process over again. Since the battery status is embedded into all 66-bit code word transmissions, if an alarm, learn or test condition is activated, the counter variable will also be reset to zero.

A test or learn transmission is initiated if switch S1 is depressed. The learn sequence will be recognized if the base station is placed in the learn mode. In either case, the switch closure wakes the PIC12C508A from sleep. The PIC12C508A then decodes the inputs and asserts the proper signals on the S0 and S1 pins of the HCS200.

With any RF link, noise is an issue that must be considered. There are some ways to control transmission integrity such as error detection/correction algorithms, repeated transmissions (simplex mode - one way) or with high end systems, the master queues each sensor for a transmission (half-duplex). The system described in this application note is configured for the simplex mode of operation and implements repeated signal transmissions for alarm conditions.

As the number of sensor modules installed in the home increases, 14 possible with this design, the odds increase that two or more sensors may converge in time for initiating a transmission cycle. The result would be a RF signal collision at the receiver and most likely all data would be lost. Once this condition occurs and since the time base for each system is not at the exact same frequency, they will typically diverge until the next occurrence.

The time base for the sensor module is the PIC12C508A which is clocked internally by the on-chip RC oscillator operating at ~ 4 MHz.

While the sensor module initiates up to four different RF transmission cycles, the most important one is the alarm condition. If the PIC12C508A detects an alarm condition, repeated RF transmissions are sent to ensure the base station receives the alarm signal. In the event that an 'OKAY' signal transmission from sensor module A and an alarm transmission from sensor module B occur at the exact same time, the alarm transmission will be received because of repeated alarm transmissions. The 'OKAY' signal only sends 1 code word transmission, while the alarm condition results in up to 5 code words transmitted.

The simple RF circuit implemented in the sensor module is an Amplitude Shift Keying (ASK) type consisting of a Colpitts oscillator with a SAW resonator. The resonator provides for a stable resonant frequency of 433.92 MHz (See Appendix A, Figure 12).

The PWM data output of the HCS200 encoder is applied to the base of the Colpitts oscillator and therefore amplitude modulates the carrier by turning the carrier on/off. The data rate is typically 833 bps.

SENSOR MODULE BATTERY CAPACITY CALCULATIONS

Before the expected battery life of the sensor module can be calculated, an operational cyclic time period must be defined. The cyclic period for the sensor module is composed of three distinct operational states: sleep, housekeeping and intentional radiation. These three states repeat on a continual basis, therefore, creating an operational cyclic profile. The profile is then used to calculate the battery capacity requirements.

For the sensor module, the cyclic time interval is approximately 1.5 hours. During this 1.5 hours, the PIC12C508A is placed in sleep 54 times. Of these 54 times, the processor wakes-up from sleep 53 times to perform some minor housekeeping and on the 54th wake-up from sleep, the intentional radiation state is executed. This is the "OK" transmission.

There is also the power-up state. This state is only executed once (initial power-up), and exhibits no significant impact on the overall battery life.

The active times for each of these states is defined below. The processor wake-up time from sleep (typical 400μs) is included in the two wake-up states.

Timing states known:

- Sleep state - typical 100 seconds (each occurrence)
- Housekeeping state – typical 56 ms (each occurrence)
- Intentional radiation state – typical 700 ms (each occurrence)

Therefore, cyclic time period is:

= (54 x 100s) + (53 x 56mS) + 700mS

= 5403.7 seconds

= 1.5010 hours

Current consumption variables known:

Sleep state current consumption:

- 3.2µA @ 6.4V_{DC} (new batteries)
- 2.0µA @ 3.3V_{DC} (battery EOL)

Housekeeping state current consumption:

- 0.70mA @ 6.4V_{DC}
- 0.30mA @ 3.3V_{DC}

Intentional radiation state current consumption:

- 4.64mA peak @ 6.4V_{DC}
- 2.22mA peak @ 3.3V_{DC}

With these operational parameters known, we can now calculate the expected battery life respective to new battery voltage. It should be noted that this is the worst case scenario and the actual battery capacity required may be less.

Calculate As Follows:

1. Calculate the percentage of time spent in each state relative to the overall cyclic time period.

Sleep state%:

- (5400s/5403.7s) x 100 = 99.932%

Housekeeping state%:

- (2.9680s/5403.7s) x 100 =.054925%

Intentional radiation state%:

- (700mS/5403.7s) x 100 =.012954%

2. Calculate the number of hours in 18 and 24 months.
   - 18 months (547.5 days) x 24hrs/day = 13,140 hours
   - 24 months (730 days) x 24hrs/day = 17,520 hours
3. With the hours, percentages and current variables known the battery capacity required for the sensor module can be developed.

For 18 months:

Sleep state:

- 13,140 hours x 99.932% x 3.2uA = 42mAh

Housekeeping state:

- 13,140 hours x .054925% x .70mA = 5.052mAh

Intentional radiation state:

- 13,140 hours x .012954% x 4.64mA = 7.899mAh

Total battery capacity required = 54.95mAh

For 24 months:

Sleep state:

- 17,520 hours x 99.932% x 3.2uA = 56.02mAh

Housekeeping state:

- 17,520 hours x .054925% x .70mA = 6.736mAh

Intentional radiation state:

- 17,520 hours x .012954% x 4.64mA = 10.530mAh

Total battery capacity required = 73.29mAh

From these calculations, we can see that if the desired operational life of the sensors is 1.5 years, the battery capacity would need to be ~55mAh. It is noted that these calculations do not take into consideration the operational characteristics of the batteries such as leakage and self discharge.

BATTERY CHARGER/ACCESSORY UNIT

The battery charger/accessory unit provides for system back-up power in the event of primary power loss. Approximately 10 hours of system operation is provided with battery operation.

This unit also contains some system peripheral circuitry and is divided into 4 main components:

1. Single stage constant voltage (constant potential) battery charger.
2. Enclosure door tamper switch feedback.
3. External piezo siren drive.
4. System remote arm/disarm using existing garage door opener. (currently not fully implemented)

Theory of Operation

The single stage battery charger consists of an adjustable voltage regulator, U1, operational amplifiers (op-amp) U2 and U3, P-channel MOSFET Q4, NPN transistor Q6, current limit resistor R15, and Schottky diode D1 (See Appendix A, Figure 9 and Figure 10). The battery used in this system is a NP4-12 Yuasa-Exide lead acid type.

The standby (float) service is a battery operational state where a constant voltage is maintained on the battery, until the battery is called on to discharge.

In this system, a constant voltage (constant potential) charging circuit is implemented to generate this maintenance voltage. The manufacturer recommends a 2.3 volts/cell maintenance voltage during this float mode. This equates to a total maintenance voltage requirement of 13.8 volts. In the event of a deep discharge cycle, the initial charging current could approach 8 amps (2CA). For this application, the initial charging current is limited to approximately 630 mA. (.16CA) When charging at 2.30 volts/cell, charging current at the final stage of charging will drop to as little as 0.002CA.

During the charge cycle, the charge current will decrease and the battery voltage will increase. When the battery voltage approaches 12.75V_{DC}, the current limit resistor will be switched out of the charge loop through turning on Q6. This will shorten the remaining battery recovery time.

NP batteries are designed to operate in standby service for approximately 5 years based upon a normal service condition, in which float charge voltage is maintained between 2.25 and 2.30 volts per cell in an ambient temperature of approximately 20°C (68°F).

In general, to assure optimum battery life, a temperature compensated charger is recommended. If the operational temperature range is between 5°C to 40°C (41°F to 104°F), it is not necessary to provide a temperature compensation function. If a temperature compensated charger is not implemented, the manufacturer recommends the maintenance voltage be set to a voltage which closely reflects the average ambient temperature based upon a compensation factor of -3mV/°C for standby (float) use.

For example:

Standard center point voltage temperature is:

- 13.8 volts @ 20°C.

Estimated average temperature is:

- 29.44°C (~85°F).

Compensated charging voltage is:

- 13.8 volts + (-3mV (29.44° - 20°)) = 13.772 volts.

For this design, the battery maintenance voltage is set to 13.77 volts. Adjustable voltage regulator, U1, is adjusted to approximately 14.00V_{DC}. This voltage accounts for the forward voltage drop of diode, D1.

This charging circuit is operating in an open loop configuration in the sense that the regulator output is manually set. If a voltage trim is required, potentiometer R18, must be adjusted.

In order to provide for some feedback to the base panel controller, a differential amplifier is configured with op-amps, U2 and U3. This amplifier configuration is such that a reference 5.1V_{DC} zener voltage is subtracted from the battery voltage. This difference is amplified and routed to TB1, pin 7. The PIC16C77 base station controller will periodically sample this voltage. If this voltage falls outside the required battery maintenance voltage, then the PIC16C77 will indicate such on the LCD and an adjustment will be required.

An airflow fan is implemented in the accessory enclosure to dissipate any gases generated by the battery and provide for moderate enclosure cooling. Air inlet and exhaust ports are provided in the enclosure. The steel enclosure dimensions are 10Hx8Wx4D. The fan current draw is approximately 100 mA @ 12V_{DC}.

For added security, an enclosure door tamper switch is utilized. If the enclosure door is opened, the PIC16C77 will be notified at pin RA2 and an alarm sequence is initiated, if the system is armed. This switch is interfaced to TB5, pins 1 and 2. While the enclosure door is closed, the feedback signal developed across R3 is active low, else if the enclosure door is opened, an active high signal is observed across R3.

In the event that an alarm condition has been initiated, the base station PIC16C77 controller will turn on NPN transistor, Q1. When Q1 is on, its collector junction will be switched to ground, and this state will turn on Q3. The drain channel of Q3 is connected to TB2, pin 1 through a 56-ohm/10W current limit resistor for direct connection to the external piezo siren. The siren implemented operates at 12V_{DC}, typically, with a current draw of ~170mA while exhibiting a ~116dBm sound pressure level annunciation. This circuit can be easily modified to allow for additional current draw should a louder siren be desired.

An eight conductor overall shielded cable provides the interface link between the base station panel and the battery charger/accessory unit.

This system also allows for an existing garage door system to arm and disarm the security system. (This feature is not completely implemented at this time)

REGULATORY CONSIDERATIONS

While low power wireless products do not need to be individually licensed, they are subject to regulation. Before low power wireless systems can be marketed in most countries, they must be certified to comply with specific technical regulations. In the U.S., the FCC issues certification. In the U.K., it is DTI, in Germany it is the FTZ, and so on.

FCC Compliance

It is noted here that Microchip Technology Incorporated does not guarantee compliance with any FCC or other regulatory requirements for this home security system, although FCC guidelines were followed and adhered to, when possible. It is the responsibility of the designer to ensure that the design is compliant to local standards.

SUMMARY

Automobile, Home or Office. All aspects of today's daily life require security. Consumers have a key pad in their hand, a security keypad on their wall and a smart card to get in the door.

The KEELOQ family of patented code hopping devices has quickly become the world standard for security applications by providing a simple yet highly secure solution for remote control locking devices, house keys, garage door openers, and home security.

From the low-cost, low-end HCS200 encoder to the high-end HCS410 encoder and transponder, Microchip's KEELOQ code hopping solutions incorporate high security, a small package outline, and low cost - an ideal combination for multifunctional, unidirectional remote keyless entry (RKE) systems. For logical and physical access control applications, such as cellular phones and smart cards, the KEELOQ family offers convenience and security in one package.

Microchip provides a complete security solution with a full range of encoders and decoders that incorporate the Company's patented KEELOQ code hopping algorithm, allowing you to get the most advanced security technology for practically a steal.

As with all security systems, it is important that the end user understand the level of security, which is required for the assets you are wanting to protect. The strength of the security system is only as strong as the weakest link. Microchips KEELOQ Security devices are proven not be a weak link.

Note:	Information contained in the application note regarding device applications and the like is intended through suggestion only and may be superseded by updates. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise.

GLOSSARY OF TERMS

ASK	Amplitude Shift Keying
EEPROM	Electrically Erasable Programmable Read Only Memory
Encryption	Method by which if "plain text" is known and the keying variables are known the "cipher text" can be produced
KEELOQ Algorithm	Nonlinear algorithm for generation of "cipher text"
JFET	Junction Field Effect Transistor
LAN	Local Area Network
LCD	Liquid Crystal Display
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
PICmicro	Microchip Technology Microcontroller
PUT	Programmable Unijunction Transistor
PWM	Pulse Width Modulation
RKE	Remote Keyless Entry
RF	Radio Frequency

REFERENCE MATERIAL

1. Application Manual, *Yuasa-Exide Incorporated, 1996.*
2. Handbook of Batteries, *2nd Edition, McGraw-Hill, David Linden, 1995.*
3. Secure Data Products Handbook, Microchip Technology Inc., *Document # DS40168, 1997.*
4. Embedded Control Handbook, Microchip Technology Inc., *Document # DS00092, 1997.*
5. PIC16C7X Data Sheet, *Document # DS30390, 1997.*
6. FCC Code of Federal Regulations, Title 47 - Telecommunication, Chapter I - Federal Communication Commission, Part 15 - Radio Frequency Devices, (http://frwebgate.access.gpo.gov/cgi-bin/multidb.cgi).

APPENDIX A: SYSTEM SCHEMATICS

FIGURE 5: *Base Station Panel (1 of 4)*

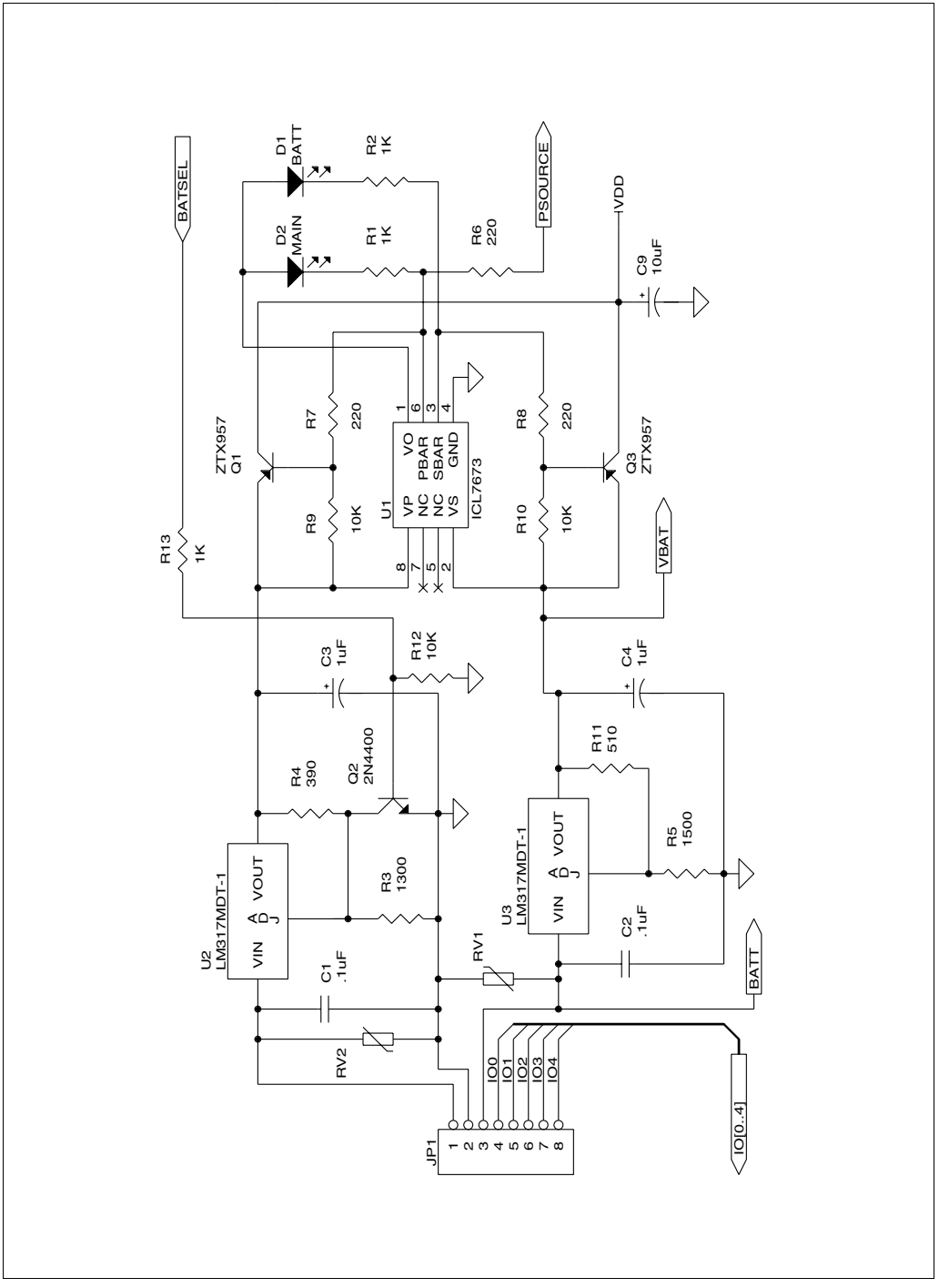


FIGURE 6: *Base Station Panel (2 of 4)*

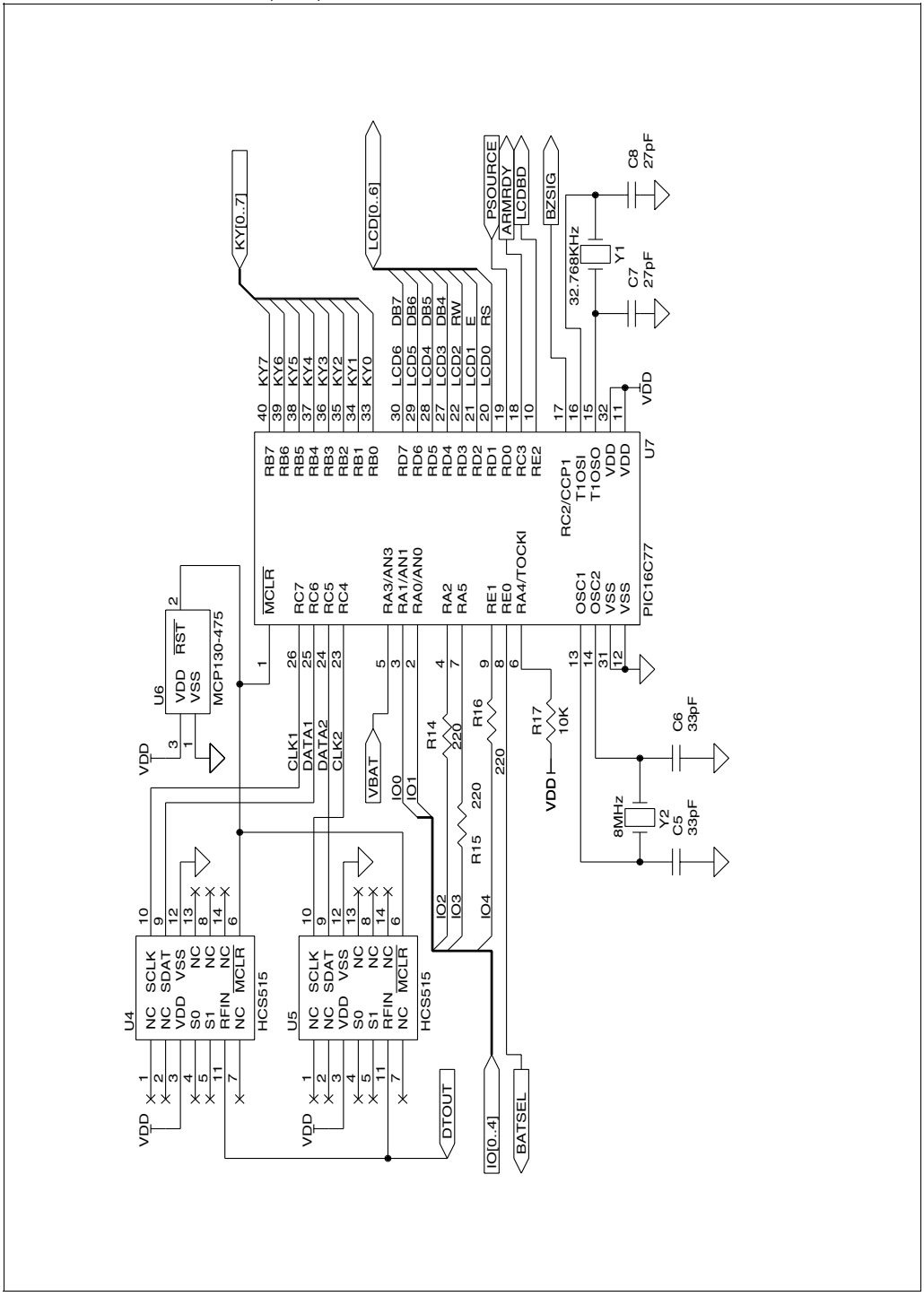


FIGURE 7: *Base Station Panel (3 of 4)*

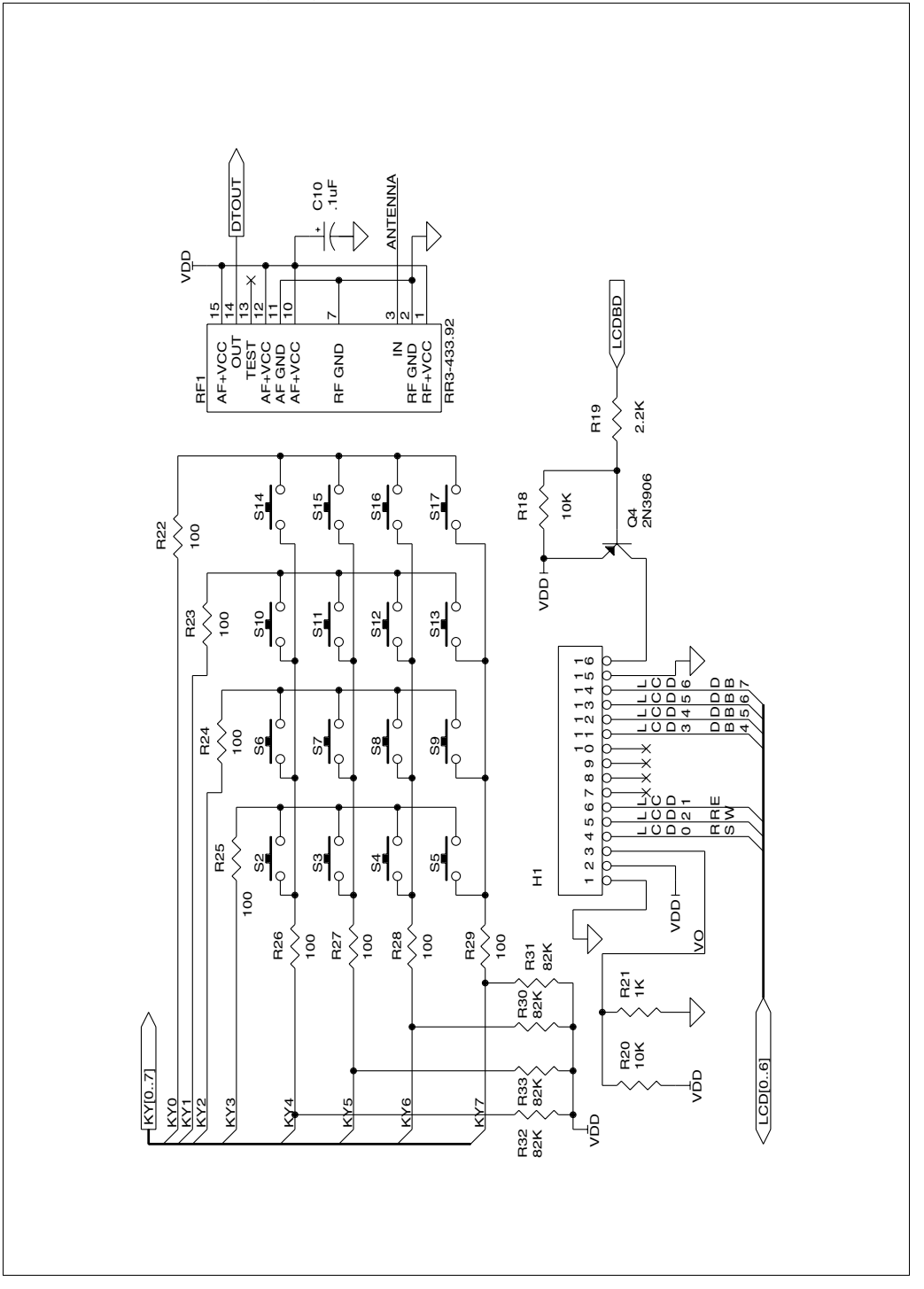


FIGURE 8: *Base Station Panel (4 of 4)*

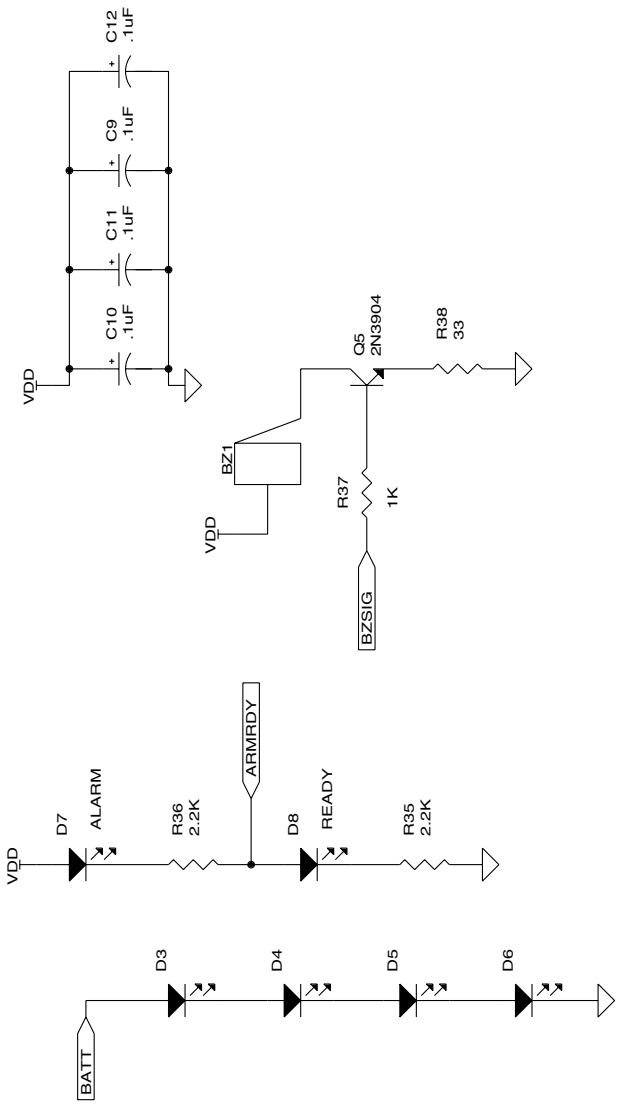


FIGURE 9: *Battery Charger/Accessory Panel (1 of 2)*

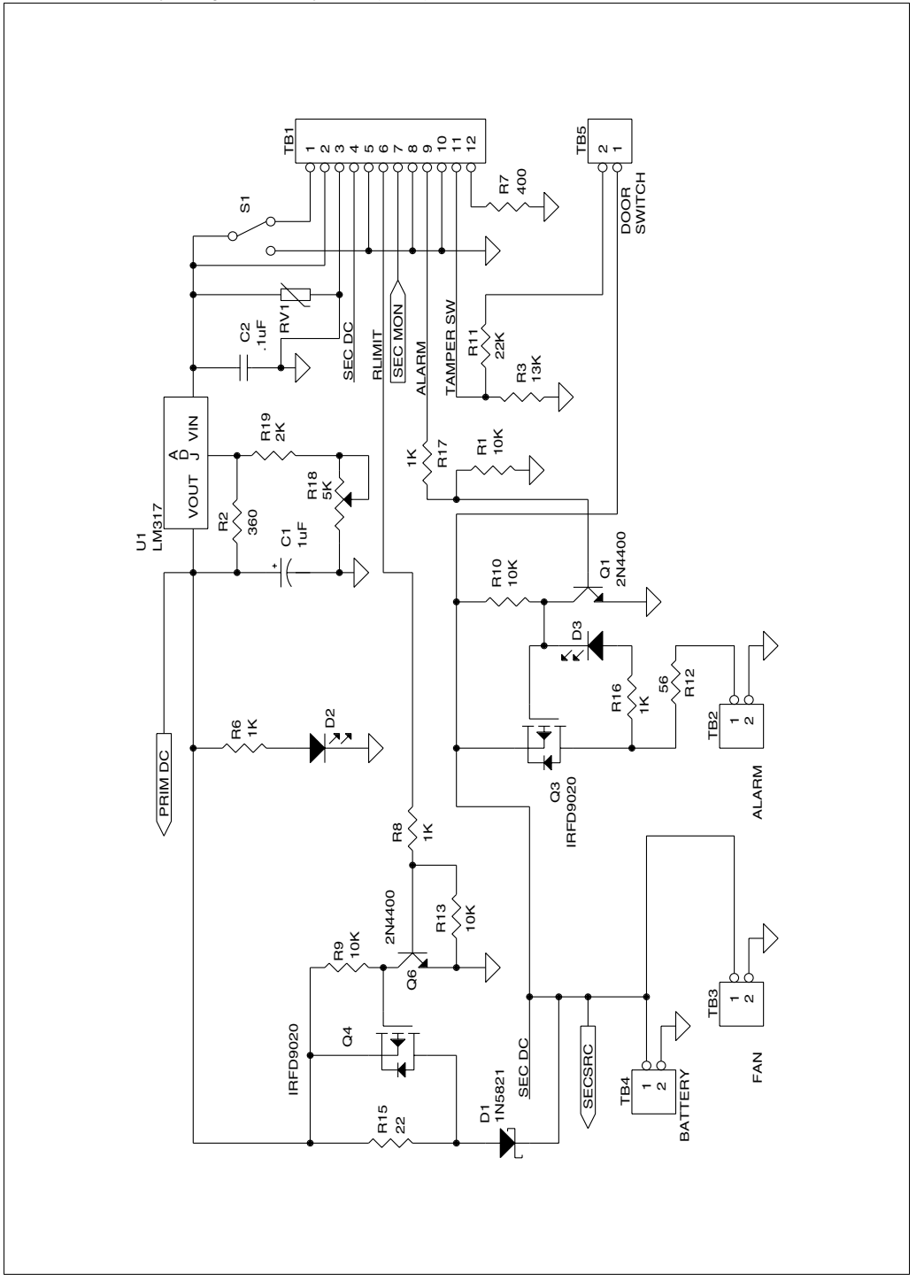


FIGURE 10: *Battery Charger/Accessory Panel (2 of 2)*

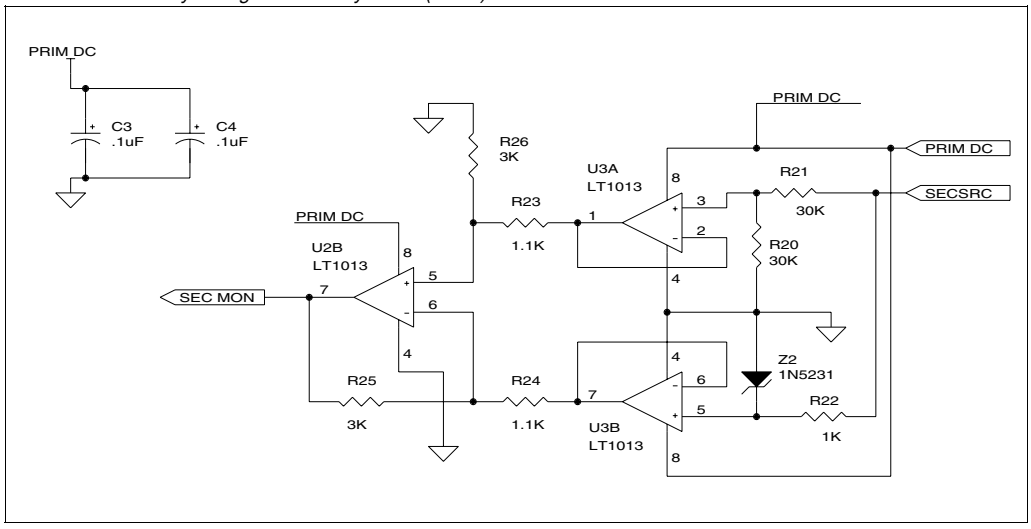


FIGURE 11: *Sensor Module (1 of 2)*

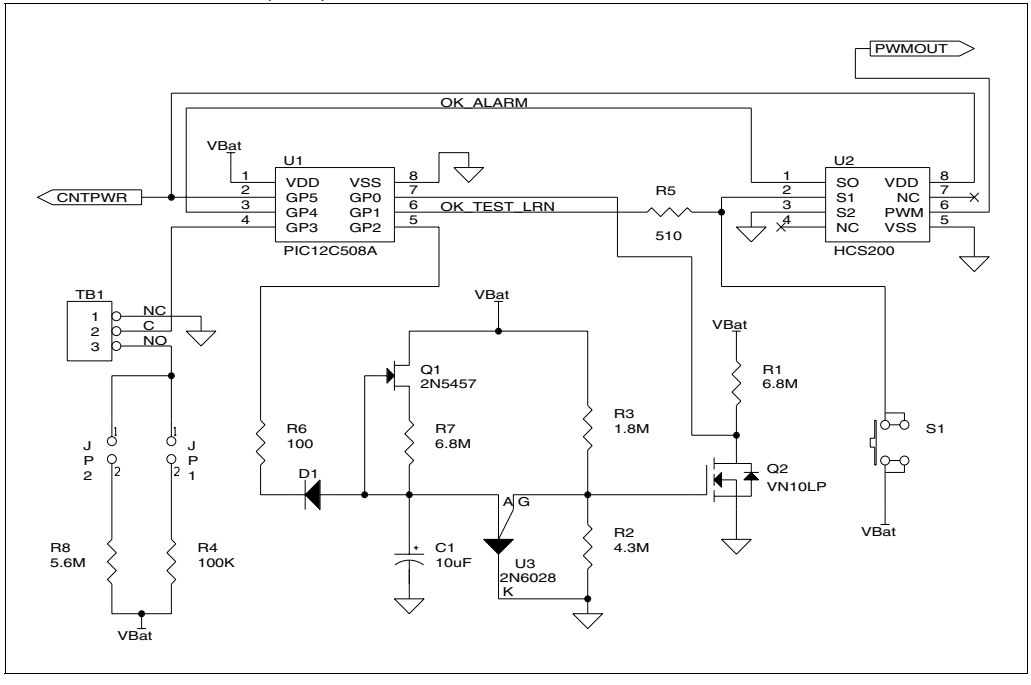
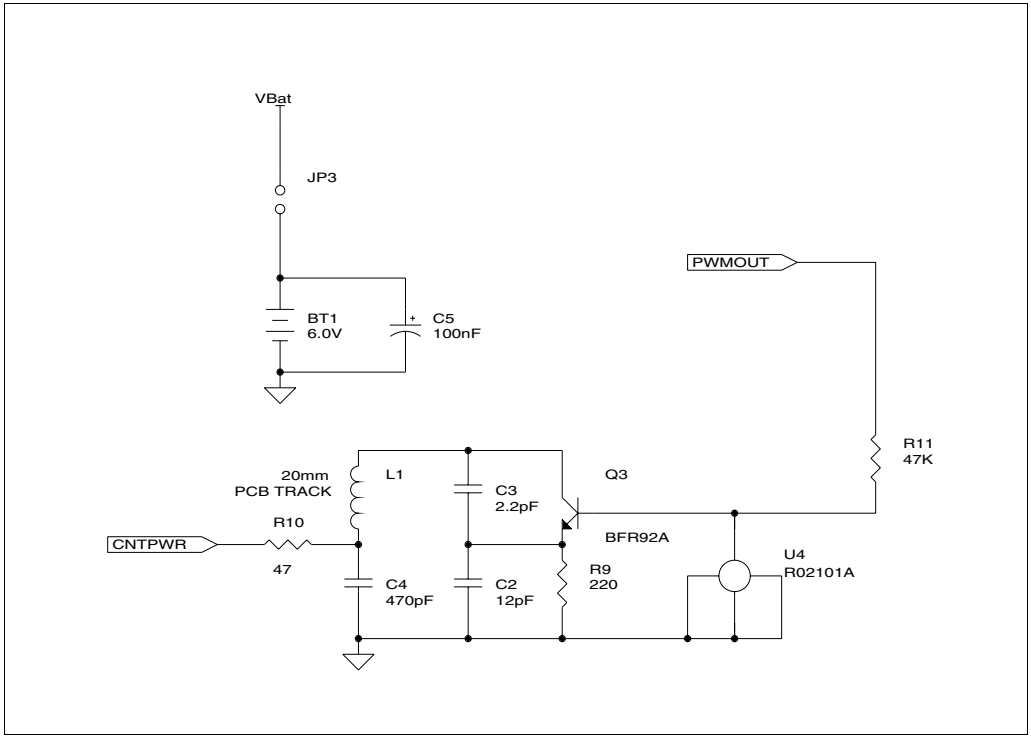


FIGURE 12: *Sensor Module (2 of 2)*



Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code

APPENDIX B: BASE STATION CODE FILES

```
/************************************************************************
*                                                                       *
*   Wireless Home Security with Keeloq and the PICmicro                 *
*                                                                       *
************************************************************************
*                                                                       *
*   Filename:       base77.c                                            *
*   Date:           07/18/99                                            *
*   File Version:   1.00                                                *
*                                                                       *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                    *
*                                                                       *
*   Author:         Richard L. Fischer                                  *
*   Company:        Microchip Technology Incorporated                   *
*                                                                       *
************************************************************************
*                                                                       *
*   System files required:                                              *
*                                                                       *
*                   powrup77.as  (Hi-Tech file, modified)               *
*                   basecode.as                                         *
*                   base77.c                                            *
*                   newmprnt.c   (Hi-Tech file)                         *
*                   baselcd.c                                           *
*                   hcsdec.c                                            *
*                   prockey.c                                           *
*                   timeset.c                                           *
*                   proctran.c                                          *
*                   diagfunc.c                                          *
*                   zonename.c                                          *
*                   codesel.c                                           *
*                   init.c                                              *
*                   delays.c                                            *
*                                                                       *
*                                                                       *
*                   pic.h        (Hi-Tech file)                         *
*                   sfr.h        (Hi-Tech file)                         *
*                   stdio.h      (Hi-Tech file)                         *
*                   string.h     (Hi-Tech file)                         *
*                   cnfig77.h                                           *
*                   base77.h                                            *
*                   baselcd.h                                           *
*                   hcsdec.h                                            *
*                   prockey.h                                           *
*                   time.h                                              *
*                   proctran.h                                          *
*                   diagfunc.h                                          *
*                   zonename.h                                          *
*                   code.h                                              *
*                   hcs515ee.h                                          *
*                                                                       *
*                                                                       *
************************************************************************
*                                                                       *
*   Notes:                                                              *
*                                                                       *
*   Device Fosc -> 8.00MHz external crystal                             *
*   Timer1 -> 32.768KHz external watch crystal                          *
*   WDT -> off                                                          *
*   Brownout -> off                                                     *
*   Powerup timer -> on                                                 *
*   Code Protect -> all                                                 *
*                                                                       *
*   Interrupt sources -                                                 *
*                     1.  4x4 Keypad on PortB                           *
*                     2.  Real-time clock - Timer1 (1sec. interval)     *
*                     3.  Timer0 (32mS interval)                        *
*                                                                       *
*                                                                       *
*                                                                       *
* Memory Usage Map:                                                     *
*                                                                       *
* User segment  $1FFA - $1FFE  $0005 (   5) bytes total User segment    *
* Program ROM   $0000 - $0002  $0003 (   3) words                       *
* Program ROM   $0004 - $180F  $180C (6156) words                       *
* Program ROM   $1AC7 - $1FF9  $0533 (1331) words                       *
* Program ROM   $2007 - $2007  $0001 (   1) words                       *
```



```
*                             $1D43 (7491) words total Program ROM  *
*                                                                   *
*                                                                   *
* Bank 0 RAM    $0020 - $0062  $0043 (  67) bytes                   *
* Bank 0 RAM    $0064 - $0069  $0006 (   6) bytes                   *
* Bank 0 RAM    $0070 - $007D  $000E (  14) bytes                   *
*                             $0057 (  87) bytes total Bank 0 RAM   *
*                                                                   *
* Bank 1 RAM    $00A0 - $00D6  $0037 (  55) bytes total Bank 1 RAM   *
* Bank 0 Bits   $0318 - $031A  $0003 (   3) bits  total Bank 0 Bits  *
*                                                                   *
*                                                                   *
*                                                                   *
********************************************************************/


    #include <pic.h>              // processor if/def file
    #include <stdio.h>
    #include "cnfig77.h"          // configuration word definitions
    #include "base77.h"           // function prototypes, defines.
    #include "hcs515ee.h"         // HCS515 EE user memory map

  __CONFIG   ( CONBLANK & BODEN_OFF & PWRTE_ON & CP_ALL & WDT_OFF & HS_OSC );


/*******************************************************************

  MAIN PROGRAM BEGINS HERE

*******************************************************************/

void main(void)
{
    Init_Adc();                           // initiliaze ADC module
    Init_Pwm();                           // initialize PWM for internal piezo use
    Init_Timer1();                        // initialize Timer1 module
    Init_Timer0();                        // initialize Timer0
    Init_Portb();                         // initialize PortB for panel keypad
    Init_Lcd();                           // initialize panel LCD module
    Init_EE_Memory();                     // initialize HCS515 EE memory sections

    flag1.battery_off = 1;                // set initial state of flag
    flag1.buzzer = 1;                     // set buzzer default state
    flag1.new_day = 1;                    // set initial flag for indicating new day
    key_wait_limit = SEC4;                // set initial key wait time (4 seconds)

    printf(" Home Security");             // display initial message line 1
    Line_2();                             // position lcd cursor on line2 / position 1
    printf("Keeloq Security");            // display initial message line 2

    decoder2_ee[0] = CLEAR;               //
    temp = Write_User_EE2( ALRM_STAT, &decoder2_ee[0], 1 );//write alarm status (clear) byte to EE

    PORTB;                                // dummy read
    RBIF = 0;                             // ensure PORTB change flag is cleared
    PEIE = 1;                             // set peripheral enable bit
    GIE = 1;                              // set global interrupt enable bit


    while( 1 )                            // main program loop
    {
        if ( HCSDATA2 )                   // test if HCS515 initiated activity
        {
            Read_Decoder_Trans();         // process reception from HCS515
        }

        if ( flag1.keyread )              // housekeeping for processing keypad entry
        {
            Process_Key();                // process key entry
            flag1.keyread = 0;            // reset flag
        }

        if ( flag1.learn_entered )        // learn activated for decoder 2
        {
            Read_Learn( 2 );              // read two bytes of learn response
            if ( ( decoder2[0] & 0xFD )  == 0x84 ) // test if learn was successful
            {
                Zone_Name();              // assign name to sensor module (zone)
            }
            flag1.learn_entered = 0;      // reset flag
        }
```

```
        if ( flag1.time_update )              // housekeeping for realtime clock?
        {
            Display_Time();                   // display and update TOD on LCD
        }

        if ( ( flag1.read_battery )  && ( !PSOURCE ) )  // battery voltage requires checking?
        {
            Test_Batt_Volt();                 // check once per hour
        }                                     // flag set in interrupt every hour

        Check_Battery_Time();                 // test if battery source was cycled on/off

        if ( flag1.arm_countdown == 1 )
        {
            Home_It();                        // set lcd cursor to line1/position 1
            printf( "Armed Countdown!" );     // format message for LCD

            if  ( time_to_arm == 0x00 )
            {
                flag2.alarm_set1= 1;          // set alarm state entry flag1
                ARMRDY = 0;                   // turn on base panel ARMED LED
                flag2.alarm_set2= 1;          // set alarm state entry flag2
                flag1.arm_countdown = 0;      // reset flag so as not to come into loop again
                flag1.arm_now = 1;            // set flag to indicate system is now ARMED
            }
        }

        if ( flag1.arm_now == 1 )             // test flag if system is ARMED now
        {
            Home_It();                        // set lcd cursor to line1/position 1
            printf("  System Armed  ");       // format message for LCD
        }

        if ( ( flag2.sensor_batt_low == 1 )  &&  ( seconds < 1 ) )
        {
            Sound_Piezo( 1 );                 // toggle internal piezo for 100mS
            Delay_100mS( 1 );                 // short delay
        }

        if ( TAMPER_SW )                      // test if accessory panel door is opened
        {
            if ( flag1.arm_now == 1 )         // is system ARMED?
            {
                ALARM_ON;                     // accessory panel door open and alarm mode set
            }
                if ( seconds < 1 )            // allow small for internal buzzer to sound
            {
                Sound_Piezo( 2 );             // toggle internal piezo for 200mS
                Delay_100mS( 1 );             // short delay
            }
        }

        if ( GARAGE_EN )                      // test for garage door open/close state change
        {
            NOP();                            // no code written/tested at this time

        }
    }
}


void mystartup( void )
{
    PORTA = 0b000000;                         // enable battery current limit, disable external
                                              // alarm
    porta_image = 0b000000;
    TRISA = 0b111001;                         // set RA1 as an output
    PORTC = 0b00001000;                       // powerup init code
    TRISC = 0b01100011;                       // RC0/1/5/6 inputs, all else outputs
    TRISD = 0b11111111;                       // ensure TRISD is set for inputs
    PORTE = 0b100;                            // LCD backdrive off
    TRISE = 0b00000010;                       // RE2/RE0 output, RE1 input

    asm( "ljmp start" );                      // return control back to program
}


void Delay_10mSb( char loop_count )           // approximate 10mS base delay
{
    unsigned int inner;                       // declare integer auto variable
    char outer;                               // declare char auto variable
```



```
    while ( loop_count )                    // stay in loop until done
    {
        for ( outer = 9; outer > 0; outer-- )
            for ( inner = 249; inner > 0; inner-- );
        loop_count--;
    }
}


void interrupt piv_isr( void )
{
    if ( T0IE && T0IF )                     // has Timer0 overflow event occurred?
    {
        if ( HCSDATA2 )                     // test for Keeloq decoder activity
        {
            key_wait = SEC4 +1;             // set key_wait to expiration time
            valid_key = ESC;                // set valid key for ESCape character
        }
        key_wait ++;                        // update key wait timer
        T0IF = 0;                           // reset Timer0 overflow flag
    }

    else if ( TMR1IE && TMR1IF )            // has Timer1 overflow event occurred?
    {
        if ( seconds < 59 )                 // is cummulative seconds < 59?
        {
            seconds++;                      // yes, so increment seconds
        }
        else                                // else seconds => 59
        {
            if ( flag1.arm_countdown )      // is countdown to ARM system flag set?
            {
                time_to_arm --;             // yes, so decrement time to arm count
            }

            seconds = 0x00;                 // reset seconds
            if ( minutes < 59 )             // is cummulative minutes < 59?
            {
                minutes++;                  // yes, so updates minutes
            }
            else                            // else minutes => 59
            {
                minutes = 0x00;             // reset minutes
                flag1.read_battery = 1;     // set flag for reading battery voltage
                if ( hours < 23 )           // is cummulative hours < 23
                {
                     hours ++;              // yes, so update hours
                }
                else
                {
                       hours = 0x00;        // reset time
                       flag1.new_day = 1;   // set flag to indicate new day
                }
            }
        }

    TMR1H |= 0x80;                          // reset Timer1 period for 1 second
    TMR1IF = 0;                             // reset Timer1 overflow flag
    }

else if ( RBIE && RBIF )                    // test for PORTB change event?
{
    Delay_10mSb( 2 );                       // 10mS delay
    switch ( valid_key = ( PORTB | 0x0F ) )
    {
        case ( 0x7F ):                      // test for single key in row 4
            key_index = 0x0C;               // 'ESC', '*', '0', '#'
            break;
        case ( 0xBF ):                      // test for single key in row 3
            key_index = 0x08;               // 'ALT', '7', '8', '9'
            break;
        case ( 0xDF ):                      // test for single key in row 2
            key_index = 0x04;               // 'PANIC', '4', '5', '6'
            break;
        case ( 0xEF ):                      // test for single key in row 1
            key_index = 0x00;               // 'AUX', '1', '2', '3'
            break;
        default:
            key_index = 0x10;               // no valid "single" key entered
            flag1.keyread = 0;              // set keyread processing flag to false
            break;
    }
```

```
    if ( key_index != 0x10 )                // if row = 1-4 valid
    {
        PORTB = 0xFF;                       // initialize PORTB outputs to logic 1's
            portb_image = 0b11101111;       // initialize mask byte for column detect
            PORTB;                          // initialize PORTB input conditions
            RBIF = 0;                       // reset interrupt flag
            portb_image >>= 1;              // rotate mask value 1 position right

            while ( CARRY )
            {
                PORTB = portb_image;        // write key selection mask value to PORTB
                NOP();                      // small settling time for output drive
                if ( RBIF )                 // is change on PORTB flag set?
                {
                       flag1.keyread = 1;   // set keyread processing flag to true
                       flag1.keyhit = 1;    // set valid key hit flag
                       CARRY = 0;           // reset carry flag
                }
                else                        // no change on PORTB so ..
                {
                         key_index++;       // increment key index
                         portb_image >>= 1; // update PORTB selection mask value
                }
            }
        }

        PORTB = 0xF0;                       // reset PORTB drive states
        if ( flag1.keyhit == 1 )            // test if there was valid key hit
        {
            valid_key = keypad[key_index];  // obtain selected key
            flag1.keyhit = 0;               // invalid key hit
        }

        PORTB;                              // PortB dummy read
        RBIF = 0;                           // reset flag
    }
}


void Display_Time( void )
{
    Line_2();                               // position lcd cursor on line2 / position 1
    printf("Time-> %02u:%02u:%02u" ,hours,minutes,seconds );
}


void putch( char data )
{
    Write_Lcd_Data( data );                 // write data to LCD via "printf"
}
```

```
/*********************************************************************
*                                                                    *
*   Filename:       base77.h                                         *
*   Date:           07/18/99                                         *
*   File Version:   1.00                                             *
*                                                                    *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                 *
*                                                                    *
*********************************************************************/


// ROM ARRAY STORAGE DEFINED HERE


#define   ESC           0x1B                // text sub. for Escape key
#define   ALT           0x40                // text sub. for Alternate key
#define   AUX           0x41                // text sub. for Auxillary key
#define   PANIC         0x50                // text sub. for Panic key

const char keypad[17] = {PANIC,'1','2','3',
                         ALT,'4','5','6',
                         AUX,'7','8','9',
                         ESC,'*','0','#','?'};


// FUNCTION PROTOTYPES

/* Functions defined in file base77.c */
void Display_Time( void );                   // function for displaying TOD
void Delay_10mSb( char loop_count);          // 10mS delay used for keypad debounce

/* Functions defined in file init.c */
extern void Init_Adc( void );                // reference linkage to defined function
extern void Init_Pwm( void);
extern void Init_Timer1( void );
extern void Init_Timer0( void );
extern void Init_Portb( void );
extern void Init_EE_Memory( void );

/* Functions defined in file baselcd.c */
extern void Init_Lcd( void );                // reference linkage to defined function
extern void Write_Lcd_Data (char data );
extern void Home_Clr( void );
extern void Home_It( void );
extern void Line_2( void );

/* Functions defined in file delays.c */
extern void Delay_100mS( char loop_count );  // reference linkage to defined function
extern void Delay_10mS( char loop_count );
extern void Delay_1mS( char loop_count );

/* Functions defined is file diagfunc.c */
extern void Sound_Piezo( char ontime );      // reference linkage to defined function
extern void Test_Batt_Volt( void );
extern void Check_Battery_Time( void );

/* Function defined in file prockey.c */
extern void Process_Key (void );             // reference linkage to defined function

/* Functions defined in file hcsdec.c */
extern void Read_Trans( char length );       // reference linkage to defined function
extern char Read_Learn( char length );
extern void Read_Decoder_Trans( void );
extern char Write_User_EE2( char address, char * wrptr, char length );

/* Function defined in file proctran.c */
extern void Process_Trans( void );           // reference linkage to defined function

/* Function defined in file zonename.c */    // reference linkage to defined function
extern void Zone_Name( void );



// VARIABLES ( DEFINED HERE )

struct event_bits1                           // bit structure for housekeeping flags
      {
        unsigned new_day        :1;          // flag for indicating new day
        unsigned arm_countdown  :1;          // flag set when counting down to arming system
        unsigned buzzer         :1;          // flag set when piezo buzzer will sound when called
        unsigned read_battery   :1;          // flag used for read battery voltage
        unsigned battery_on     :1;          // flag set when battery source is selected
```

```
        unsigned battery_off     :1;          // flag set when battery is not selected
        unsigned time_update     :1;          // flag used for updating LCD and E2 Memory
        unsigned erase_all       :1;          // flag set when HCS515 erase all operation complete
        unsigned battery_sel     :1;          // flag set when battery source is selected
        unsigned erase_entered   :1;          // flag set when HCS515 erase all operation entered
        unsigned arm_now         :1;          // flag set when system is immediately armed w/o user
                                              // code
        unsigned learn_entered   :1;          // flag set when HCS515 learn operation entered
        unsigned code_valid      :1;          // flag set after user security access code accepted
        unsigned code_entered    :1;          // flag set when user security access code entered
        unsigned keyread         :1;          // flag set for indicating keypad selection needs
                                              // processed
        unsigned keyhit          :1;          // flag set when valid key hit on 4x4 keypad is
                                              // detected
    } flag1;                                  // variable name

struct event_bits2                            // define bit structure for housekeeping flags
    {
        unsigned                 :1;          // bit padding
        unsigned alarm_set1      :1;          // flag set when system is armed ( 1 of 2 )
        unsigned                 :6;          // bit padding
        unsigned valid_rcv       :1;          // flag used if HCS515 demod data is correct
        unsigned sensor_batt_low :1;          // flag indicating if sensor module battery is low
        unsigned                 :5;          // bit padding
        unsigned alarm_set2      :1;          // flag set when system is armed ( 2 of 2 )
    } flag2;                                  // variable name


char porta_image;                             // define PortA image register
char portb_image;                             // define PortB image register
char key_index;                               // define key index variable for const array
char hours;                                   // define variable used for hours
char minutes;                                 // define variable used for minutes
char seconds;                                 // define variable used for seconds

bank1 char temp;
bank1 char key_wait;                          // define variable for key wait timer
bank1 char valid_key;                         // define variable for 4x4 keypad value
bank1 char key_wait_limit;                    // define variable for key wait timer


// VARIABLES ( REFERENCE DECLARATION )

//extern bank1 char decoder1[10];             // reference linkage, array storage for valid trans.
                                              // reception (decoder 1)
extern bank1 char decoder2[10];               // reference linkage, array storage for valid trans.
                                              // reception (decoder 2)
//extern bank1 char decoder1_ee[6];           // reference linkage to array variable
extern bank1 char decoder2_ee[6];             // reference linkage to array variable
extern bank1 char time_to_arm;                // reference linkage to defined variable


// PORTBITS ( DEFINED HERE )

#define PortBit(port,bit)    ((unsigned)&(port)*8+(bit))

static bit TAMPER_SW  @  PortBit(PORTA,5);    // Input from Accessory unit door tamper switch

static bit ARMRDY     @  PortBit(PORTC,3);    // Alarm/Ready Light bias control
static bit HCSCLK2    @  PortBit(PORTC,4);    // HSC515 clock input (Ref. U5)
static bit HCSDATA2   @  PortBit(PORTC,5);    // HCS515 data output (Ref. U5)
static bit HCSDATA1   @  PortBit(PORTC,6);    // HCS515 data output (Ref. U4)
static bit HCSCLK1    @  PortBit(PORTC,7);    // HCS515 clock input (Ref. U4)
static bit PSOURCE    @  PortBit(PORTD,0);    // Power Source indication status

static bit GARAGE_EN  @  PortBit(PORTE,1);    // Input from Existing garage door receiver
static bit LCDBD      @  PortBit(PORTE,2);    // LCD Back Drive on/off


// MACROS ( DEFINED HERE )

#defineNOP()asm(" nop")                       // define NOP macro

#define ALARM_ON       porta_image |= 0b000100;\
                       PORTA = porta_image;  // enable external alarm

#define ALARM_OFF      porta_image &= ~0b000100;\
                       PORTA = porta_image;  // enable external alarm

#define   SEC4           120                  //
```

```
/************************************************************************
*                                                                       *
*   Wireless Home Security with Keeloq and the PICmicro                 *
*                                                                       *
************************************************************************
*                                                                       *
*   Filename:       baselcd.c                                           *
*   Date:           07/18/99                                            *
*   File Version:   1.00                                                *
*                                                                       *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                    *
*                                                                       *
*   Author:         Richard L. Fischer                                  *
*   Company:        Microchip Technology Incorporated                   *
*                                                                       *
************************************************************************
*                                                                       *
*   Files required:                                                     *
*                                                                       *
*                   baselcd.h                                           *
*                                                                       *
************************************************************************
*                                                                       *
*   Notes: The routines within this file are required for               *
*          communicating with the system LCD (2x16) module              *
*          connected to PORTD.                                          *
*                                                                       *
*                                                                       *
*   PORTD: RD4 - RD7 ( 4-bit data interface )                           *
*          RD3 -     ( R/W control signal )                             *
*          RD2 -     ( E control signal )                               *
*          RD1 -     ( RS control signal )                              *
*                                                                       *
*   PORTE: RE2 -     ( LCD backlight on/off / 60mA draw )               *
*                                                                       *
*                                                                       *
************************************************************************/

    #include "baselcd.h"                    // function prototypes, defines..


void Init_Lcd( void )                       // initialize LCD display
{
    CONTROL = 0x01;                         // initial state of control lines
    TRISCTRL = 0x01;                        // initialize control lines

    Delay15ms();                            // ~15mS delay upon powerup

    DATA = 0x30;                            // output setup data to LCD
    E = 1;                                  // set enable high
    NOP();
    E = 0;                                  // set enable low

    Delay5ms();                             // ~5mS delay

    DATA = 0x30;                            // output setup data to LCD
    E = 1;                                  // set enable high
    NOP();
    E = 0;                                  // set enable low

    Delay200us();                           // ~200uS delay

    DATA = 0x30;                            // output setup data to LCD
    E = 1;                                  // set enable high
    NOP();
    E = 0;                                  // set enable low
    Lcd_Busy();                             // test for lcd_busy state

    DATA = 0x20;                            // output setup data to LCD
    E = 1;                                  // set enable high
    NOP();
    E = 0;                                  // set enable low
    Lcd_Busy();                             // test for lcd_busy state

    Write_Lcd_Cmd( 0x28 );                  // define 4 bit interface, 2 lines. 5x7 dots
    Write_Lcd_Cmd( 0x0C );                  // display on, cursor on, blink off
    Write_Lcd_Cmd( 0x01 );                  // clear display
    Write_Lcd_Cmd( 0x06 );                  // entry mode set..
    Write_Lcd_Cmd( 0x28 );                  //
}
```

```
void Write_Lcd_Cmd( char cmd )                // subroutiune for lcd commands
{
    DATA = ( cmd & 0xF0 );                    // send upper 4 bits of command
    E = 1;                                    // set enable high
    NOP();
    E = 0;                                    // set enable low
    DATA = ( ( cmd << 4 ) & 0xF0 );           // now send lower 4 bits of command
    E = 1;                                    // set enable high
    NOP();
    E = 0;                                    // set enable low
    Lcd_Busy();                               // check lcd busy flag
}


void Write_Lcd_Data( char data )              // subroutine for lcd data
{
    DATA = 0x00;                              // set pins to defined state
    RS = 1;                                   // assert register select to 1
    DATA |= ( data & 0xF0 );                  // send upper 4 bits of data
    E = 1;                                    // set enable high
    NOP();                                    //
    E = 0;                                    // set enable low
    DATA &= 0x0F;
    DATA |= ( data << 4 );                    // now send lower 4 bits of data
    E = 1;                                    // set enable high
    NOP();                                    //
    E = 0;                                    // set enable low
    RS = 0;                                   // negate register select to 0
    Lcd_Busy();                               // check lcd busy flag
}


void Lcd_Busy( void )
{
    TRISDATA_7 = 1;                           // make line an input
    RW = 1;                                   // assert R/W for read operation

    while( TRUE )                             // stay in loop until lcd not busy
    {
      E = 1;                                  // set enable high
      NOP();                                  // ensure tDDR spec is met before test
      if ( !DATA_7 )                          // is busy bit negated
      {
        E = 0;                                // set enable low
        RW = 0;                               // negate R/W for write operation
        TRISDATA_7 = 0;                       // return line to output
        return;                               // exit busy routine
      }
      else
      {
        E = 0;                                // set enable low
        NOP();
        E = 1;                                // set enable high
        NOP();
        E = 0;                                // set enable low
      }
    }
}


void Delay15ms( void )                        // approximate 15ms delay
{
    char outer, inner;
    for (outer = 24; outer > 0; outer--)
        for (inner = 250; inner > 0; inner--);
}


void Delay5ms( void )                         // approximate 5ms delay
{
    char outer, inner;
    for (outer = 8; outer > 0; outer--)
        for (inner = 253; inner > 0; inner--);
}


void Delay200us( void )                       // approximate 200us delay
{
    char delay;
        for (delay = 66; delay > 0; delay--);
}
```



```
void Cursor_Right( void )                   // shift lcd cursor right 1 position
{
    Write_Lcd_Cmd( 0x14 );
}

void Cursor_Left( void )                    // shift lcd cursor left 1 position
{
    Write_Lcd_Cmd( 0x10 );
}

void Display_Shift( void )                  // shift lcd display contents
{
    Write_Lcd_Cmd (0x1C );
}

void Home_Clr( void )                       // clear lcd and set cursor to line1/position 1
{
    Write_Lcd_Cmd( 0x01 );
}

void Home_It( void )                        // set lcd cursor to line1/position 1
{
    Write_Lcd_Cmd( 0x02 );
}

void Line_2( void )                         // clear lcd and set to line2/position 1
{
    Write_Lcd_Cmd (0xC0 );
}
```

```
/************************************************************************
*                                                                       *
*   Filename:       baselcd.h                                           *
*   Date:           07/18/99                                            *
*   File Version:   1.00                                                *
*                                                                       *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                    *
*                                                                       *
************************************************************************/


// FUNCTION PROTOTYPES

/* Functions defined in file baselcd.c */
void Write_Lcd_Cmd( char cmd );              // write command to lcd
void Lcd_Busy( void );                       // busy flag check
void Delay15ms( void );                      // approximate 15ms delay
void Delay5ms( void );                       // approximate 5ms delay
void Delay200us( void );                     // approximate 200us delay

static unsigned char PORTD @ 0x08;           // reference/declare of PORTD variable
static unsigned char TRISD @ 0x88;           // reference/declare of TRISD variable


// MACROS ( DEFINED HERE )

#define  TRUE      1
#define  NOP() asm(" nop")

#define  CONTROL   PORTD                     // Port for lcd control lines
#define  TRISCTRL  TRISD                     // I/O setup for control Port
#define  DATA      PORTD                     // Port for lcd data
#define  TRISDATA  TRISD                     // I/O setup for data Port


// PORTBITS ( DEFINED HERE )

#define PortBit(port,bit)    ((unsigned)&(port)*8+(bit))

static bit DATA_7     @ PortBit(PORTD,7);    // DATA bit 7 for LCD busy check
static bit TRISDATA_7 @ PortBit(TRISD,7);    // TRIS bit 7 for LCD busy check
static bit RW         @ PortBit(PORTD,3);    // R/W control bit for LCD
static bit RS         @ PortBit(PORTD,1);    // Register select bit for LCD
static bit E          @ PortBit(PORTD,2);    // Enable/Clock for bit for LCD
```



```
/************************************************************************
*                                                                       *
*   Wireless Home Security with Keeloq and the PICmicro                 *
*                                                                       *
************************************************************************
*                                                                       *
*   Filename:       delays.c                                            *
*   Date:           07/18/99                                            *
*   File Version:   1.00                                                *
*                                                                       *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                    *
*                                                                       *
*   Author:         Richard L. Fischer                                  *
*   Company:        Microchip Technology Incorporated                   *
*                                                                       *
************************************************************************
*                                                                       *
*   Files required:                                                     *
*                                                                       *
*                   pic.h                                               *
*                                                                       *
*                                                                       *
************************************************************************
*                                                                       *
*   Notes: The delay routines within this file are used                 *
*          by other system functions.                                   *
*                                                                       *
*                                                                       *
************************************************************************/

    #include <pic.h>                        // processor if/def file


#define   NOP() asm(" nop")                 // define NOP macro


void Delay_1S( char loop_count )            // approximate 1S base delay
{
    unsigned int  inner;                    // define/declare auto type int variable
    unsigned char outer;                    // define/declare auto type char variable

    while ( loop_count )
    {
        for ( outer = 145; outer > 0; outer-- )
            for ( inner = 985; inner > 0; inner-- )
            {
                NOP();
                NOP();
                NOP();
                NOP();
                NOP();
            }
        loop_count--;                       // decrement loop iteration counter
    }
}


void Delay_100mS( char loop_count )         // approximate 100mS base delay
{
    unsigned int  inner;                    // define/declare auto type int variable
    unsigned char outer;                    // define/declare auto type char variable

    while ( loop_count )
    {
        for ( outer = 87; outer > 0; outer-- )
            for ( inner = 255; inner > 0; inner-- );

        loop_count--;                       // decrement loop iteration counter
    }
}


void Delay_10mS( char loop_count )          // approximate 10mS base delay
{
    unsigned int  inner;                    // define/declare auto type int variable
    unsigned char outer;                    // define/declare auto type char variable

    while ( loop_count )
    {
        for ( outer = 9; outer > 0; outer-- )
            for ( inner = 246; inner > 0; inner-- );
```

```
        loop_count--;                           // decrement loop iteration counter
    }
}


void Delay_1mS( char loop_count )               // approximate 1mS base delay
{
    unsigned int  inner;                        // define/declare auto type int variable
    unsigned char outer;                        // define/declare auto type char variable

    while ( loop_count )
    {
        for ( outer = 1; outer > 0; outer-- )
            for ( inner = 219; inner > 0; inner-- );

        loop_count--;                           // decrement loop iteration counter
    }
}


void Delay_200uS( void )                        // approximate 200us delay
{
    char delay;                                 // define/declare auto type char variable
        for ( delay = 66; delay > 0; delay-- );
}



void Delay_20uS( char loop_count )              // approximate 20us base delay
{
    char delay;                                 // define/declare auto type char variable

    for ( ; loop_count > 0 ; loop_count-- )
        for ( delay = 6; delay > 0; delay-- );
}


void Delay_10uS(void)                           // approximate 10us delay
{
    char delay;                                 // define/declare auto type char variable

    NOP();
    NOP();
        for ( delay = 2; delay > 0; delay-- );
}


void Delay_5uS(void)                            // approximate 5us delay
{
    NOP();
    NOP();
    NOP();
    NOP();
    NOP();
}
```



```
/*****************************************************************************
*                                                                            *
*   Wireless Home Security with Keeloq and the PICmicro                      *
*                                                                            *
*****************************************************************************
*                                                                            *
*   Filename:       diagfunc.c                                               *
*   Date:           07/18/99                                                 *
*   File Version:   1.00                                                     *
*                                                                            *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                         *
*                                                                            *
*   Author:         Richard L. Fischer                                       *
*   Company:        Microchip Technology Incorporated                        *
*                                                                            *
*****************************************************************************
*                                                                            *
*   Files required:                                                          *
*                                                                            *
*                   pic.h                                                    *
*                   stdio.h                                                  *
*                   diagfunc.h                                               *
*                   hcs515ee.h                                               *
*                                                                            *
*                                                                            *
*                                                                            *
*****************************************************************************
*                                                                            *
*                                                                            *
*   Notes: The routines within this file are used for:                       *
*                                                                            *
*   1. Monitoring battery voltage and determining if current                 *
*      limiting resistor is switched in or out.                              *
*   2. Converting ADC result to a floating point number                      *
*   3. Initiating drive signal for piezo buzzer                              *
*   4. Monitoring and recording battery on/off T.O.D. and number             *
*      of on/off cycles.                                                     *
*                                                                            *
*                                                                            *
*****************************************************************************/

    #include <pic.h>
    #include <stdio.h>
    #include "diagfunc.h"
    #include "hcs515ee.h"



void Test_Batt_Volt( void )
{
    ADON = 1;                                   // power up ADC module
    NOP();                                      // short delay
    ADGO = 1;                                   // start conversion
    while( ADGO );                              // wait here until conversion complete
    ADON = 0;                                   // power down ADC module

    if ( ADRES > 0xBA )                         // is battery voltage > 12.75Vdc?
    {
        BATT_CURR_ON;                           // yes, so remove battery charge current limit
    }
    else
    {
        BATT_CURR_OFF;                          // no, so battery charge current limit on
    }
    flag1.read_battery = 0;                     // reset read battery status flag
}



void Battery_Voltage( void )
{
    Test_Batt_Volt();                           // perform conversion on scaled battery voltage
    battery_volts = ( ( ( ADRES * 0.019726562 ) / 2.7477 ) + 5.13 ) * 1.9984;
}



void Sound_Piezo( char ontime )
{
    if ( flag1.buzzer )                         // test if buzzer is enabled
    {
        GIE = 0;                                // disable global interrupt enable
        CCP1CON = 0b00001100;                   // set CCP1 for PWM
```

```
        do                                      // loop
        {
            Delay_1mS( 100 );                   // ~99.91 mS
        } while ( --ontime );                   // number of 100mS loops

        CCP1CON = 0x00;                         // turn off CCP1 module
        GIE = 1;                                // re-enable global interrupt enable
    }
}


void Check_Battery_Time( void )
{

/* Test if battery has been selected and record time */
    if ( PSOURCE  &&  !flag1.battery_on )       // test if battery is selected
    {                                           // if PSOURCE == 1, then battery is selected
        decoder2_ee[0] = hours;                 // write hours count to buffer for EE write
        decoder2_ee[1] = minutes;               // write minutes count to buffer for EE write
        decoder2_ee[2] = seconds;               // write seconds count to buffer for EE write
        temp = Write_User_EE2( BT_ON_HRS, &decoder2_ee[0], 3 ); // write battery on-time to EE
                                                // memory

        flag1.battery_on = 1;                   // set flag to indicate battery is selected
        flag1.battery_off = 0;                  // reset flag for battery off time
    }

/* Test if battery is de-selected and record time */
    if ( !PSOURCE  &&  !flag1.battery_off )     // test if line is primary source
    {                                           // if !PSOURCE == 0, main power is on
        decoder2_ee[0] = hours;                 // write hours count to buffer for EE write
        decoder2_ee[1] = minutes;               // write minutes count to buffer for EE write
        decoder2_ee[2] = seconds;               // write seconds count to buffer for EE write
        temp = Write_User_EE2( BT_OFF_HRS, &decoder2_ee[0], 3 );// write battery off-time to EE
                                                // memory

        Delay_1mS( 5 );

        temp = Read_User_EE2( BT_ON_CNT, &decoder2_ee[0], 1 );// read battery on cycle count
        decoder2_ee[0] ++;

        Delay_1mS( 5 );
        temp = Write_User_EE2( BT_ON_CNT, &decoder2_ee[0], 1 ); // write updated battery on cycle
                                                // count

        flag1.battery_off = 1;                  // set flag to indicate battery is not selected
        flag1.battery_on = 0;                   // reset battery on status flag
    }

/* Test if new 24 hour period has began */
    if ( flag1.new_day )                        // test if new day flag has been set
    {
        decoder2_ee[0] = 0x00;                  // set array element to 0
        decoder2_ee[1] = 0x00;                  // set array element to 0
        decoder2_ee[2] = 0x00;                  // set array element to 0
        decoder2_ee[3] = 0x00;                  // set array element to 0
        decoder2_ee[4] = 0x00;                  // set array element to 0
        decoder2_ee[5] = 0x00;                  // set array element to 0

        temp = Write_User_EE2( BT_ON_CNT, &decoder2_ee[0], 1 );// reset daily battery on count
        temp = Write_User_EE2( BT_ON_HRS, &decoder2_ee[0], 6 );// reset battery on/off time to 0

        flag1.new_day = 0;                      // reset 24 hour flag
    }
}
```

```
/*********************************************************************
*                                                                    *
*   Filename:       diagfunc.h                                       *
*   Date:           07/18/99                                         *
*   File Version:   1.00                                             *
*                                                                    *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                 *
*                                                                    *
*********************************************************************/



// FUNCTION PROTOTYPES

/* Functions defined in file hcsdec.c */
extern char Write_User_EE2( char address, char * wrptr, char length );
extern char Read_User_EE2( char address, char * rdptr, char length );

/* Function defined in file delays.c */
extern void Delay_1mS( char loop_count );    // reference linkage to defined function


// VARIABLES ( REFERENCE DECLARATION )

extern struct event_bits1                    // bit structure for housekeeping flags
      {
        unsigned new_day         :1;         // flag for indicating new day
        unsigned arm_countdown   :1;         // flag set when counting down to arming system
        unsigned buzzer          :1;         // flag set when piezo buzzer will sound when called
        unsigned read_battery    :1;         // flag used for read battery voltage
        unsigned battery_on      :1;         // flag set when battery source is selected
        unsigned battery_off     :1;         // flag set when battery is not selected
        unsigned time_update     :1;         // flag used for updating LCD and E2 Memory
        unsigned erase_all       :1;         // flag set when HCS515 erase all operation complete
        unsigned battery_sel     :1;         // flag set when battery source is selected
        unsigned erase_entered   :1;         // flag set when HCS515 erase all operation entered
        unsigned arm_now         :1;         // flag set when system is immediately armed w/o user
                                             // code
        unsigned learn_entered   :1;         // flag set when HCS515 learn operation entered
        unsigned code_valid      :1;         // flag set after user security access code accepted
        unsigned code_entered    :1;         // flag set when user security access code entered
        unsigned keyread         :1;         // flag set for indicating keypad selection needs
                                             // processed
        unsigned keyhit          :1;         // flag set when valid key hit on 4x4 keypad is
                                             // detected
    } flag1;                                 // variable name


extern bank1 char decoder2_ee[6];            // reference linkage to defined variable

extern bank1 char temp;                      // reference linkage to defined variable

extern char porta_image;                     // reference linkage to defined variable
extern char hours;                           // reference linkage to defined variable
extern char minutes;
extern char seconds;


// VARIABLES ( DEFINED HERE )

bank1 double battery_volts;                  // variable defined here


// PORTBITS ( DEFINED HERE )

#define PortBit(port,bit)    ((unsigned)&(port)*8+(bit))

static bit PSOURCE    @  PortBit(PORTD,0);   // Power Source indication status
static bit BATSEL     @  PortBit(PORTE,0);   // Battery Select (test only)


// MACROS ( DEFINED HERE )

#define NOP()asm(" nop")                     // define NOP macro

#define BATT_CURR_ON   porta_image |= 0b000010;\
                       PORTA = porta_image;  // enable maximum battery charge current

#define BATT_CURR_OFF  porta_image &= ~0b000010;\
                       PORTA = porta_image;  // enable maximum battery charge current
```

```
/************************************************************************
*                                                                       *
*   Wireless Home Security with Keeloq and the PICmicro                 *
*                                                                       *
************************************************************************
*                                                                       *
*   Filename:       hcsdec.c                                            *
*   Date:           07/18/99                                            *
*   File Version:   1.00                                                *
*                                                                       *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                    *
*                                                                       *
*   Author:         Richard L. Fischer                                  *
*   Company:        Microchip Technology Incorporated                   *
*                                                                       *
************************************************************************
*                                                                       *
*   Files required:                                                     *
*                                                                       *
*                   pic.h                                               *
*                   stdio.h                                             *
*                   hcsdec.h                                            *
*                                                                       *
*                                                                       *
*                                                                       *
************************************************************************
*                                                                       *
*                                                                       *
*   Notes: The routines within this file are required for               *
*          communicating with the system HCS515 decoders.               *
*                                                                       *
*                                                                       *
*                                                                       *
************************************************************************/

    #include <pic.h>
    #include <stdio.h>
    #include "hcsdec.h"


void Read_Trans( char length )
{
    TRISDATA2 = 1;                          // ensure pin direction is input
    lrn_ptr = decoder2                      // initialize pointer

    Delay_20uS( 3 );                        // intervals of 20uS delay ( 61uS total )
    HCSCLK2 = 1;                            // assert clock - provide ack to HCS515

    Delay_20uS( 2 );                        // ~42uS ( Tclh )
    HCSCLK2 = 0;                            // negate clock
    Delay_20uS( 2 );                        // ~42uS ( Tcll )

    while ( length )                        // stay while length != 0
    {
        command.bit_counter = 8;            // initialize bit counter
        do
        {
            HCSCLK2 = 1;                    // set clock high (total high time - 26uS)
            NOP();                          // add in .5uS delay
            Delay_20uS( 1 );                // ~24uS delay
            HCSCLK2 = 0;                    // set clock low (sample data after falling edge )
                                            // (total low time - 31.5uS)
            temp_buffer >> = 1;>            // rotate buffer contents towards LSB
            if ( HCSDATA2 )                 // test if data line is logic 1
            {
                temp_buffer |= 0x80;        // set bit 7 to logic 1
            }
            Delay_20uS( 1 );                // ~24uS delay

        } while ( --command.bit_counter );  // decrement counter and test if complete

        *lrn_ptr++ = temp_buffer;           // save off read byte and update pointer
        length --;                          // decrement loop length counter
    }
}


char Read_Learn( char length )
{
    TRISDATA2 = 1;                          // ensure pin direction is input
    HCSCLK2 = 0;                            // ensure clock line is low
```

```
    lrn_ptr = decoder2;                         // initialize pointer
    command.treq_timeout = 0x0000;              // initialize union variable (integer element)

    do                                          // loop until HCS515 responds
    {                                           // or until ~35 seconds expires
        Delay_1mS( 2 );                         // intervals of 2mS delay
        Delay_20uS( 20 );                       // plus
        command.treq_timeout++;                 // increment response wait timer

        if ( HCSDATA2 == 1 )                    // has HCS515 responded ?
        {
            command.treq_timeout = TREQ +1;     // if so then set timeout expiration
        }
    } while ( command.treq_timeout <= TREQ );

    Home_Clr();                                 // clear lcd and set to line1 / position 1
    if ( HCSDATA2 == 0 )                        // is data line still low?
    {                                           // transmission has not been detected 35 sec.
        return ( TREQ_ERR );                    // return with error code
    }

// At this point the first of two learn transmissions has been received
    Delay_100mS( 1 );                           // delay of 100mS delay

    if ( HCSDATA2 != 0 )                        // data line should be lo at this pointw
    {
        return ( TREQ_ERR );                    // else, return with error code
    }

    command.treq_timeout = 0x0000;              // initialize union variable (integer element)

    printf("1st trans. rcv'd" );                // format message for LCD
    Line_2();                                   // position lcd cursor on line2 / position 1
    printf("30 seconds left " );                // format message for LCD

// wait here for second transmission
    do                                          // loop until HCS515 responds
    {                                           // or until ~30sec expires
        Delay_1mS( 1 );                         // intervals of 1.3mS delay
        Delay_20uS( 20 );                       // plus more
        command.treq_timeout++;                 // increment response wait timer

        if ( HCSDATA2 == 1 )                    // has HCS515 responded ?
        {
            command.treq_timeout = TREQ +1;     // if so then set timeout expiration
        }
    } while ( command.treq_timeout <= TREQ );

    Home_Clr();                                 // clear lcd and set to line1 / position 1
    if ( HCSDATA2 == 0 )                        // is data line still low?
    {
        return ( TREQ_ERR );                    // return with error code
    }

// At this point the second and final learn transmission has been received by the HCS515
    command.tack_timeout = 0x00;                // initialize union variable (char element)
    Delay_20uS( 4 );                            // Tcla wait period (spec - 50ouS min)
    HCSCLK2 = 1;                                // assert clock, send acknowledge to HCS515

    do                                          // loop until HCS515 responds
    {                                           // or until wait time expires
        Delay_5uS();
        command.tack_timeout++;                 // increment response wait timer

        if ( HCSDATA2 == 0 )                    // has HCS515 responded ?
        {
            command.tack_timeout = TACK +1;     // if so then set timeout expiration
        }
    } while ( command.tack_timeout <= TACK );

    if ( HCSDATA2 == 1 )                        // is data line still high?
    {
        return ( TACK_ERR );                    // return with error code
    }

    Delay_20uS( 2 );                            // ~42uS ( Tclh )
    HCSCLK2 = 0;                                // negate clock
    Delay_20uS( 2 );                            // ~42uS ( Tcll )
    TRISDATA2 = 1;                              // set data pin direction

// Read x number of bytes from the second transmission ( up to 10 )
    while ( length )
    {
```

```
        command.bit_counter = 8;              // initialize bit counter
        do
        {
            HCSCLK2 = 1;                      // set clock high
            NOP();                            // add in .5uS delay
            Delay_20uS( 1 );                  // ~22uS delay
            HCSCLK2 = 0;                      // set clock low ( data sampled on falling edge )

            temp_buffer >> = 1;               // rotate towards LSB position
            if ( HCSDATA2 )                   // test if carry bit set
            {
                temp_buffer |= 0x80;          // set bit 7 to logic 1
            }
            Delay_20uS( 1 );                  // ~22uS delay

        } while ( --command.bit_counter );    // decrement counter and test if complete

        *lrn_ptr++ = temp_buffer;             // save off learn status message and update pointer
        length --;                            // decrement byte counter
    }

    return ( NO_ERROR );                      // return with no read error
}



char Learn( void )
{
    flag1.learn_entered = 0;                  // reset learn entered flag

    temp = Command_Mode( ACTIVE_LRN, DUMMY, DUMMY ); // initiate command mode
    if ( temp != 0 )
    {
        return ( temp );                      // return with error code
    }

// At this point Command Mode for Learn has been sent to HCS515, wait for acknowledge
// HCS515 should respond within 20uS (max) after clock line is asserted
    TRISDATA2 = 1;                            // ensure data direction pin state is input
    Delay_20uS( 2 );                          // wait for ~ 40uS (Tlrn-20uS(min) Tlrn )
    HCSCLK2 = 1;                              // set clock high, begin TACK period

    command.tack_timeout = 0x00;              // initialize variable
    do                                        // loop until HCS515 responds with
    {                                         // data line high or until time expires
                                              // loop time ~8uS (total: 5*8us= 40uS)
        command.tack_timeout++;               // increment timeout counter

        if ( HCSDATA2 == 1 )                  // has HCS515 responded and entered learn mode ?
        {
            command.tack_timeout = TACK_LRN +1;// if so then set timeout expiration
        }
    } while ( ( !HCSDATA2 ) && ( command.tack_timeout <= TACK_LRN ) );

    if ( !HCSDATA2 )                          // is DATA line still low after TACK
    {
        return ( TACK_LRN_ERR );              // return with error code
    }

    Delay_20uS( 1 );                          // ~22uS delay (Tresp spec 20-1000uS)
    HCSCLK2 = 0;                              // set clock low
    Delay_20uS( 1 );                          // ~22uS delay (TACK2 spec 10uS max)

    flag1.learn_entered = 1;                  // set flag learn entered mode
    return ( NO_ERROR );                      // return with no error condition
}



char Erase_All( void )
{
    flag1.erase_all = 0;                      // reset flag for erase all status

    temp = Command_Mode( ERASE_ALL, SUBCOM0, DUMMY ); // initiate command mode
    if ( temp != 0 )
    {
        return ( temp );                      // return with error code
    }

    TRISDATA2 = 1;                            // ensure data direction pin state is input
    Delay_20uS( 2 );                          // wait for ~ 42uS( Tera time )
    HCSCLK2 = 1;                              // set clock high, begin TACK period
```



```
    command.tack_timeout = 0x00;              // initialize variable
    do                                        // loop until HCS515 responds with
    {                                         // driving data line high or until time expires
        Delay_20uS( 46 );                     // wait ~900uS per loop (spec - 210mS max)
        command.tack_timeout++;               // increment timeout counter

        if ( HCSDATA2 == 1 )                  // has HCS515 finished erasing xmtrs ?
        {
            command.tack_timeout = TACK_ERASE +1; // if so then set timeout expiration
        }
    } while ( ( !HCSDATA2 ) && ( command.tack_timeout <= TACK_ERASE ) );

    if ( !HCSDATA2 )                          // is DATA line still low after TACK (max)
    {
        return ( TACK_ERASE_ERR );            // return with error code
    }

    Delay_20uS( 1 );                          // ~22uS delay (Tresp)
    HCSCLK2 = 0;                              // set clock low
    Delay_20uS( 1 );                          // ~22uS delay (TACK2 wait)

    flag1.erase_all = 1;                      // set flag to indicate all xmtrs have been erased
    return ( NO_ERROR );                      // return with no error condition
}


char Read_User_EE2( char address, char * rdptr, char length )
{
    temp = Command_Mode( READ, address, DUMMY ); // initiate command mode
    if ( temp != 0 )
    {
        return ( temp );                      // return with error code
    }

    Delay_20uS( 50 );                         // Trd time period (1000 - 2000uS)
    TRISDATA2 = 1;                            // ensure data direction pin state is input

    while ( length )                          // test byte read counter is zero
    {
        command.bit_counter = 8;              // initialize bit counter
        do
        {
            HCSCLK2 = 1;                      // set clock high
            NOP();                            // add in .5uS delay
            Delay_20uS( 1 );                  // ~22uS delay
            HCSCLK2 = 0;                      // set clock low ( data sampled on falling edge )

            temp_buffer >> = 1;               // rotate towards LSB position
            if ( HCSDATA2 )                   // test if carry bit set
            {
                temp_buffer |= 0x80;          // set bit 7 to logic 1
            }
            Delay_20uS( 1 );                  // ~22uS delay

        } while ( --command.bit_counter );    // decrement counter and test if complete

    *rdptr++ = temp_buffer;                   // save off read byte and update pointer

    Delay_20uS( 50 );                         // Trd time period (1000 - 2000uS)
    length --;                                // decrement read counter
    }

    HCSDATA2 = 0;                             // ensure data line is set low
    TRISDATA2 = 1;                            // ensure direction of pin is input
    return ( NO_ERROR );                      // return with no error
}


char Write_User_EE2( char address, char * wrptr, char length )
{
    temp = Command_Mode( WRITE, address, WRITE ); // initiate command mode
    if ( temp != 0 )
    {
        return ( temp );                      // return with error code
    }

    while ( length )                          // test for end of string (null character)
    {
        TRISDATA2 = 0;                        // ensure data direction pin state is output
        command.bit_counter = 8;              // initialize bit counter
        temp_buffer = *wrptr;                 // assign data to temp_buffer
```

```
        do
        {
            temp_buffer >> = 1;              // rotate LSB into carry
            if ( CARRY )                     // test if carry bit set
            {
                HCSDATA2 = 1;                // set data line high
            }
            else
            }
                HCSDATA2 = 0;                // set data line low
            }

            HCSCLK2 = 1;                     // set clock high
            Delay_20uS( 1 );                 // ~22uS delay
            HCSCLK2 = 0;                     // set clock low ( data sampled on falling edge )
            Delay_20uS( 1 );                 // ~22uS delay ( also provides for required Tds )

        } while ( --command.bit_counter );   // decrement counter and test if complete

        Delay_20uS( 1 );                     // wait for ~ 20uS (spec -20uS min  -Twr)
        HCSDATA2 = 0;                        // set data line low
        Delay_20uS( 1 );                     // wait for ~ 20uS
        TRISDATA2 = 1;                       // set pin direction for input
        HCSCLK2 = 1;                         // set clock high, begin TACK period

        Delay_1mS( 3 );                      // wait here 3mS to start

        command.tack_timeout = 0x40;         // initialize variable

        do                                   // loop until HCS515 responds (1 loop = 48uS)
        {                                    // or until ~9mS expires
            Delay_20uS( 2 ) ~41uS delay
            command.tack_timeout++;          // increment timeout counter

            if ( HCSDATA2 == 1 )             // has HCS515 responded ?
            {
                command.tack_timeout = TACK_WR +1;// if so then set timeout expiration
            }
        } while ( command.tack_timeout <= TACK_WR );

        Delay_20uS( 2 );                     // ~20uS delay (Tresp)
        HCSCLK2 = 0;                         // set clock low
        Delay_20uS( 1 );                     // ~20uS delay (TACK2 wait)

    length --;                               // decrement write count
    wrptr++;                                 // increment data pointer
    }

    TRISDATA2 = 1;                           // ensure data direction pin state is input
    Delay_1mS( 1 );
    return ( NO_ERROR );                     // return with no error condition
}



char Command_Mode( char decoder_command, char cmd_byte1, char cmd_byte2 )
{
    command.treq_timeout = 0x0000;           // initialize union variable (integer element)

    TRISDATA2 = 1;                           // ensure data direction pin state is input
    HCSCLK2 = 1;                             // set clock high to initiate command
    TRISCLK2 = 0;                            // ensure data direction pin state is output

    do                                       // loop until HCS515 responds (1 loop = 33uS)
    {                                        // or until ~500mS expires (total loop 660mS)
        Delay_20uS( 1 );                     // ~24uS delay
        command.treq_timeout++;              // increment response wait timer

        if ( HCSDATA2 == 1 )                 // has HCS515 responded ?
        {
            command.treq_timeout = TREQ +1;  // if so then set timeout expiration
        }
    } while ( command.treq_timeout <= TREQ );

    if ( HCSDATA2 == 0 )                     // is data line still low?
    {
        return ( TREQ_ERR );                 // return with error code
    }

// At this point the HCS515 has responded by asserting data line high
// so bring the clock low
    Delay_20uS( 2 );                         // (Tresp, spec-20uS(min)) ~42uS delay
    HCSCLK2 = 0;                             // bring clock low, ack to decoder
```



```
// At this point, HCS515 has acknowledged PICmicro by negating data line (low)
    Delay_20uS( 2 );                        // Tstart (20uS min) ~ 41.50uS delay
    TRISDATA2 = 0;                          // ensure data direction pin state is output

    command.bit_counter = 8;                // initialize bit counter
    temp_buffer = decoder_command;          // assign command to temp_buffer
    do
    {
        temp_buffer >> = 1;                 // rotate LSB into carry
        if ( CARRY )                        // test if carry bit set
        {
            HCSDATA2 = 1;                   // set data line high
        }
        else
        {
            HCSDATA2 = 0;                   // set data line low
        }

        HCSCLK2 = 1;                        // set clock high
        Delay_20uS( 1 );                    // ~23.5uS delay
        HCSCLK2 = 0;                        // set clock low( data sampled on falling edge )
        Delay_20uS( 1 );                    // ~23.5uS delay (also provides for required Tds)

    } while ( --command.bit_counter );      // decrement counter and test if complete


    command.bit_counter = 8;                // initialize bit counter
    temp_buffer = cmd_byte1;                // 1st byte after command byte to temp_buffer
    do
    {
        temp_buffer >> = 1;                 // rotate LSB into carry
        if ( CARRY )                        // test if carry bit set
        {
            HCSDATA2 = 1;                   // set data line high
        }
        else
        {
            HCSDATA2 = 0;                   // set data line low
        }

        HCSCLK2 = 1;                        // set clock high
        Delay_20uS( 1 );                    // ~20uS delay
        HCSCLK2 = 0;                        // set clock low ( data sampled on falling edge )
        Delay_20uS( 1 );                    // ~20uS delay ( also provides for required Tds)

    } while ( --command.bit_counter );      // decrement counter and test if complete

    if ( cmd_byte2 == WRITE )               // test if Write EE is using command mode function
    {
        return ( NO_ERROR );
    }

    command.bit_counter = 8;                // initialize bit counter
    temp_buffer = cmd_byte2;                // 2nd byte after command byte to temp_buffer
    do
    {
        temp_buffer >> = 1;                 // rotate LSB into carry
        if ( CARRY )    // test if carry bit set
        {
            HCSDATA2 = 1;                   // set data line high
        }
        else
        {
            HCSDATA2 = 0;                   // set data line low
        }

        HCSCLK2 = 1;                        // set clock high
        Delay_20uS( 1 );                    // ~20uS delay
        HCSCLK2 = 0;                        // set clock low( data sampled on falling edge )
        Delay_20uS( 1 );                    // ~20uS delay ( also provides for required Tds)

    } while ( --command.bit_counter );      // decrement counter and test if complete

    return ( NO_ERROR );                    // return with no error
}
```

```
/***********************************************************************
*                                                                      *
*   Filename:       hcsdec.h                                           *
*   Date:           07/18/99                                           *
*   File Version:   1.00                                               *
*                                                                      *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                   *
*                                                                      *
***********************************************************************/


// FUNCTION PROTOTYPES

/* Functions defined in file hcsdec.c */
char Learn( void );                             // reference linkage to defined function
void Read_Trans( char length );
char Erase_All( void );
char Read_User_EE2( char address, char * rdptr, char length );
char Write_User_EE2( char address, char * wrptr, char length );
char Command_Mode( char decoder_command, char cmd_byte1, char cmd_byte2 );

/* Functions defined in file baselcd.c */
extern void Home_Clr( void );                   // reference linkage to defined function
extern void Line_2( void );

/* Functions defined in file delays.c */
extern void Delay_5uS( void );                  // reference linkage to defined function
extern void Delay_20uS( char loop_count );
extern void Delay_1mS( char loop_count );
extern void Delay_100mS( char loop_count );
extern void Delay_10mS( char loop_count );


// VARIABLES ( REFERENCE DECLARATION )

extern struct event_bits1                       // bit structure for housekeeping flags
      {
        unsigned new_day          :1;           // flag for indicating new day
        unsigned arm_countdown    :1;           // flag set when counting down to arming system
        unsigned buzzer           :1;           // flag set when piezo buzzer will sound when called
        unsigned read_battery     :1;           // flag used for read battery voltage
        unsigned battery_on       :1;           // flag set when battery source is selected
        unsigned battery_off      :1;           // flag set when battery is not selected
        unsigned time_update      :1;           // flag used for updating LCD and E2 Memory
        unsigned erase_all        :1;           // flag set when HCS515 erase all operation complete
        unsigned battery_sel      :1;           // flag set when battery source is selected
        unsigned erase_entered    :1;           // flag set when HCS515 erase all operation entered
        unsigned arm_now          :1;           // flag set when system is immediately armed w/o user
                                                // code
        unsigned learn_entered    :1;           // flag set when HCS515 learn operation entered
        unsigned code_valid       :1;           // flag set after user security access code accepted
        unsigned code_entered     :1;           // flag set when user security access code entered
        unsigned keyread          :1;           // flag set for indicating keypad selection needs
                                                // processed
        unsigned keyhit           :1;           // flag set when valid key hit on 4x4 keypad is
                                                // detected
    } flag1;                                    // variable name

extern struct event_bits2                       // define bit structure for housekeeping flags
    {
        unsigned                  :1;           // bit padding
        unsigned alarm_set1       :1;           // flag set when system is armed ( 1 of 2 )
        unsigned                  :6;           // bit padding
        unsigned valid_rcv        :1;           // flag used if HCS515 demod data is correct
        unsigned sensor_batt_low:1;             // flag indicating if sensor module battery is low
        unsigned                  :5;           // bit padding
        unsigned alarm_set2       :1;           // flag set when system is armed ( 2 of 2 )
    } flag2;                                    // variable name


extern bank1 char temp;// reference linkage to defined variable


// VARIABLES ( DEFINED HERE )

bank1 char * bank1 lrn_ptr;                     // define pointer
bank1 char decoder1[10];                        // array storage for valid trans.
                                                // reception (decoder 1)
bank1 char decoder2[10];                        // array storage for valid trans.
                                                // reception (decoder 2)
```

```
bank1 char decoder1_ee[6];                  // array for reading/writing to HCS515 EE
bank1 char decoder2_ee[6];                  // array for reading/writing to HCS515 EE

bank1 char temp_buffer;                     // temp buffer for writing/reading data ( HCS515 )

bank1 union notify {
       unsigned int treq_timeout;           // integer element
       unsigned int tack_wr_timeout;        // integer element
       unsigned char tresp_timeout;         // char element
       unsigned char tack_timeout;          // char element
       unsigned char bit_counter;           // char element
} command;                                  // variable defined


// PORTBITS ( DEFINED HERE )

#define PortBit(port,bit)    ((unsigned)&(port)*8+(bit))

static bit HCSDATA1   @  PortBit(PORTC,6);  // declare bit for HCS515 data line
static bit HCSDATA2   @  PortBit(PORTC,5);  // declare bit for HCS515 data line
static bit HCSCLK1    @  PortBit(PORTC,7);  // declare bit for HCS515 clock line
static bit HCSCLK2    @  PortBit(PORTC,4);  // declare bit for HCS515 clock line

static bit TRISDATA1  @  PortBit(TRISC,6);  // declare bit for HCS515 data pin direction
static bit TRISDATA2  @  PortBit(TRISC,5);  // declare bit for HCS515 data pin direction
static bit TRISCLK1   @  PortBit(TRISC,7);  // declare bit for HCS515 clock pin direction
static bit TRISCLK2   @  PortBit(TRISC,4);  // declare bit for HCS515 clock pin direction


// MACROS DEFINED HERE

#defineNOP()asm(" nop")                     // define NOP macro


// DEFINE HCS515 DECODER COMMANDS

#define READ        0xF0                    // Read a byte from the user
#define WRITE       0xE1                    // Write a byte to the user
#define ACTIVE_LRN  0xD2                    // Activate a learn sequence
#define ERASE_ALL   0xC3                    // Activate an erase all
#define SUBCOM0     0x00                    // Erase all Subcommand byte
#define SUBCOM1     0x01                    // Erase all Subcommand byte


#define TREQ         20000                  //
#define TACK  50                            //
#define TACK_WR        240
#define TACK_LRN         5                  // timeout constant for Learn (Tack)
#define TACK_ERASE     250

#define NO_ERROR         0
#define TREQ_ERR         1
#define TACK_ERR         2
#define TACK_WR_ERR      4
#define TACK_LRN_ERR    10
#define TACK_ERASE_ERR  20

#define DUMMY         0x81
```

```
/************************************************************************
*                                                                       *
*   Wireless Home Security with Keeloq and the PICmicro                 *
*                                                                       *
************************************************************************
*                                                                       *
*   Filename:       init.c                                              *
*   Date:           07/18/99                                            *
*   File Version:   1.00                                                *
*                                                                       *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                    *
*                                                                       *
*   Author:         Richard L. Fischer                                  *
*   Company:        Microchip Technology Incorporated                   *
*                                                                       *
************************************************************************
*                                                                       *
*   Files required:                                                     *
*                                                                       *
*                   pic.h                                               *
*                                                                       *
************************************************************************
*                                                                       *
*   Notes: The routines within this file are required for               *
*          initializing PICmicro peripherals and the HCS515 EE.         *
*                                                                       *
*                                                                       *
************************************************************************/

    #include <pic.h>                        // processor if/def file


extern char Write_User_EE2( char address, char * wrptr, char length );
extern const char zone_name_address[];      // 89,90,97,9E,A5,AC,B3
extern bank1 char decoder2_ee[6];
extern bank1 char temp;



void Init_Adc( void )
{
    ADCON1 = 0b101;                         // RA0/RA1 analog, RA3 Vref
    ADCON0 = 0b10000000;                    // Tosc32, channel 0
}


void Init_Pwm( void )
{
    PR2 = 122;                              // set for frequency of ~4KHz
    CCPR1L = 40;                            // duty cycle ~33%
    T2CON = 0b00000001;                     // set TMR2 prescale for 4
    TMR2ON = 1;                             // turn on TMR2
}


void Init_Timer1( void )
{
    TMR1CS = 1;                             // Timer1 clock select, external
    T1OSCEN = 1;                            // enable Timer1 oscillator mode
    TMR1L = 0x00;
    TMR1H = 0x80;                           // initialize timer1
    TMR1IF = 0;                             // reset Timer1 overflow flag
    TMR1IE = 1;                             // enable TMR1 Overflow interrupt
    TMR1ON = 1;                             // turn on Timer1 module
}


void Init_Timer0( void )
{
    OPTION = 0b11010111;                    // set Timer0 for 1:256, internal clock
    TMR0 = 0x00;                            // set Timer0 for initial state
    T0IF = 0;                               // reset Timer0 overflow flag
    T0IE = 1;                               // enable Timer0 Overflow interrupt
}


void Init_Portb( void )
{
    PORTB = 0b11110000;                     // PortB setup
    TRISB = 0b11110000;                     // RB7-RB4 inputs, RB3-RB0 outputs
    PORTB;                                  // dummy read of PortB
    RBIF = 0;                               // reset flag
```



```
    RBIE = 1;                                   // set PortB interrupt on change
}


void Init_EE_Memory( void )
{
    char index = 0;                             // define auto type variable
    decoder2_ee[0] = 0x00;                      // set array element to known state
    decoder2_ee[1] = 0x00;                      // set array element to known state
    decoder2_ee[2] = 0x00;                      // set array element to known state
    decoder2_ee[3] = 0x00;                      // set array element to known state

    do
    {
        temp = Write_User_EE2( ( ( zone_name_address[index] ) + 1 ), &decoder2_ee[0], 4 );
        index++;
    } while ( index < 7 );
}
```

```
/************************************************************************
*                                                                       *
*   Wireless Home Security with Keeloq and the PICmicro                 *
*                                                                       *
************************************************************************
*                                                                       *
*   Filename:       prockey.c                                           *
*   Date:           07/18/99                                            *
*   File Version:   1.00                                                *
*                                                                       *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                    *
*                                                                       *
*   Author:         Richard L. Fischer                                  *
*   Company:        Microchip Technology Incorporated                   *
*                                                                       *
************************************************************************
*                                                                       *
*   Files required:                                                     *
*                                                                       *
*                   pic.h                                               *
*                   stdio.h                                             *
*                   string.h                                            *
*                   prockey.h                                           *
*                   hcs515ee.h                                          *
*                                                                       *
************************************************************************
*                                                                       *
*   Notes: The routines within this file are required for               *
*          processing the 4x4 keypad selections.                        *
*                                                                       *
*   This file generates ~ 650 words of code. If additional              *
*   requirements are required, the file may need to be reduced into     *
*   4 smaller separate nodes.                                           *
*                                                                       *
*   Possible: 1 node for PANIC Key                                      *
*             1 node for ALTnerate Key                                  *
*             1 node for AUXiliary Key                                  *
*             1 node for ESCape Key                                     *
*                                                                       *
*   It is also noted here that there are many tasks generated           *
*   within this file which include user menus. The menus can be         *
*   reduced or removed and still retain system functionality            *
*   and operation. This approach would also reduce the overall          *
*   code size of this node. However, the menus do help with user        *
*   interaction.                                                        *
*                                                                       *
************************************************************************/

    #include <pic.h>                          // processor if/def file
    #include <stdio.h>
    #include <string.h>
    #include "prockey.h"                      // function prototypes, variables, defines..
    #include "hcs515ee.h"



void Process_Key( void )
{
    int i;                                    // define auto type integer variable

    switch ( valid_key )
    {
        case ( PANIC ):                       // if 'PANIC' key has been pressed

        key_wait = 0x00;                      // reset key wait timer
        while ( ( valid_key != ESC ) && ( key_wait < key_wait_limit ) )
        {
            if ( valid_key == '*' )           // labled '*'  on keypad
            {
/* Arm System here w/o entry of User Code and w/o "ARM" time delay */
                flag2.alarm_set1= 1;          // set alarm state entry flag1
                ARMRDY = 0;                   // turn on base panel ARMED LED
                Home_It();                    // set lcd cursor to line1/position 1
                printf("  System Armed  ");   // format message for LCD
                flag2.alarm_set2= 1;          // set alarm state entry flag2
                flag1.arm_now = 1;            // set immediate alarm entry flag true
                key_wait = SEC4 + 1;          // set key wait timer to expire
                flag1.arm_countdown = 0;      // reset flag for ARMED countdown
            }
```

```
/* Arm System here with entry of User Code and enable "ARM" time delay */
           else if ( ( valid_key ==  '#' )  &&  ( !flag1.arm_now ) )
           {
               codestring[4] = 0x00;          // initialize variable
               csindex = 0;                   // initialize array index
               flag1.keyread = 0;             // reset key read flag

               Home_Clr();                    // clear lcd and set cursor to line1/position 1
               printf( "To Arm... Enter" );   // format message for LCD
               Line_2();                      // set lcd cursor to line2/position 1
               printf( "User Code-> " );      // format message for LCD

               Code_Select();                 // function for entering Master/User code

               temp = Read_User_EE2( USER_CODE, &tempstring[0], 4 ); // read User Code from EE

               Home_Clr();                     // clear lcd and set cursor to line1/position 1
               if ( ( i = strcmp( codestring, tempstring ) )  < 0  )
               {
               flag1.code_valid = 0;          // flag set false, invalid Master Code entered
               }
           else if ( i > 0 )
               {
               flag1.code_valid = 0;          // flag set false, invalid Master Code entered
               }
           else
               {
               printf("Armed Countdown!");    // format message for LCD
               Sound_Piezo( 1 );              // 100ms enable of internal piezo
               flag1.code_valid = 1;          // flag set true, valid master code entered
               flag1.arm_countdown = 1;       // set flag to indicate ARM countdown state
               time_to_arm = 6;               // set time to arm for ~ 6 minutes
           }

           if ( !flag1.code_valid )           // test if there was a Invalid Master Code entry
           {
               printf( "* Invalid User *" ); // format message for LCD
               Line_2();                      // set lcd cursor to line2/position 1
               printf( "* Code Entered *" ); // format message for LCD
               Sound_Piezo( 1 );              // 100ms enable of internal piezo
               Delay_100mS( 1 );              // short delay
               Sound_Piezo( 1 );              // 100ms enable of internal piezo
               Delay_100mS( 6 ) ;             // short delay for user to view LCD message
               Home_Clr();                    // clear lcd and set cursor to line1/position 1
           }
       }

       else if ( valid_key ==  '1' )          // labled 1 on keypad
       {
           flag1.buzzer = 1;                  // set flag so piezo buzzer will sound
           key_wait = SEC4 + 1;               // set key wait timer to expire
       }
   }
   break;


   case ( ALT ):                              // if 'ALTernate' key has been pressed

   key_wait = 0x00;// reset key wait timer
   while ( ( valid_key != ESC ) && ( key_wait < key_wait_limit ) )
   {
       if ( valid_key == TEST )               // TEST == 5 on keypad (labeled TEST)
       {
           Home_Clr();                        // clear lcd and set cursor to line1/position 1
           printf( "Battery.. Sel[1]" );      // format message for LCD
           Line_2();                          // set lcd cursor to line2/position 1
           printf( "Vdc[2],  Time[3]" );      // format message for LCD

           key_wait = 0x00;                   // reset key wait timer
           while ( ( valid_key != ESC ) && ( key_wait < key_wait_limit ) )
           {
               if ( valid_key == '1' )        // is battery select option chosen?
               {
                 if ( LCDBD )                 // test if backlight is on
                 {
                   BATSEL ^= 1;               // toggle battery backup select
                   flag1.battery_sel ^= 1;    // set flag indicating battery is selected
               }
               key_wait = SEC4 + 1;           // set key wait timer to expire
           }
```

```
        else if ( valid_key == '2' )      // is battery voltage option chosen?
        {
            while ( valid_key != ESC )    // stay in loop until ESCape key is pressed
            {
                Home_It();                // set lcd cursor to line1/position 1
                if ( !PSOURCE )           // test if primary power is on
                {
                    Battery_Voltage();    // perform conversion on ADC channel 0
                    printf("Battery Voltage:" ); // format message for LCD
                    Line_2();             // set lcd cursor to line2/position 1
                    printf("   %.3f Vdc   ", battery_volts ); // format message for LCD
                }
                else                      // primary is not on
                {
                    printf(" Battery Source " ); // format message for LCD
                    Line_2();             // set lcd cursor to line2/position 1
                    printf("  is selected   " ); // format message for LCD
                }
                Delay_100mS( 1 );         // ~100 mS delay
            }
        }

        else if ( valid_key == '3' )      // display recorded battery on time
        {
            Home_Clr();                   // clear lcd and set cursor to line1/position 1
            temp = Read_User_EE2( BT_ON_HRS, &decoder2_ee[0], 6 ); // read recorded time of
                                          // battery on
            printf( "on:  -> %02u:%02u:%02u " , decoder2_ee[0],decoder2_ee[1],decoder2_ee[2] );
            Line_2();                     // set lcd cursor to line2/position 1
            printf( "off:--> %02u:%02u:%02u" , decoder2_ee[3],decoder2_ee[4],decoder2_ee[5] );
            valid_key = 0x20;             // reset key
            while ( valid_key != ESC );   // stay in loop until ESCape key is pressed
            Home_Clr();                   // clear lcd and set cursor to line1/position 1
            printf( "Battery daily  " );  // format message for LCD
            Line_2();                     // set lcd cursor to line2/position 1
            temp = Read_User_EE2( BT_ON_CNT, &decoder2_ee[0], 1 ); // read daily battery cycle
                                          // count

            printf( "cycle count= %02u" , decoder2_ee[0] ); // format message for LCD
            valid_key = 0x20;             // reset key
            while ( valid_key != ESC );   // stay in loop until ESCape key is pressed
        }
    }
}

else if ( valid_key == ALARM_STATUS )     // ALARM_STATUS == 7 on keypad (labeled INSTANT)
{
    Home_It();                            // set lcd cursor to line1/position 1
    temp = Read_User_EE2( ALRM_HRS, &decoder2_ee[0], 4 ); // read alarm status and time info

    if ( decoder2_ee[3] == CLEAR )
    {
        printf( " No Zone Fault! " );     // format message for LCD
    }

    else if ( decoder2_ee[3] == ALRM )
    {
        printf( "Fault:  %02u:%02u:%02u", decoder2_ee[0],decoder2_ee[1],decoder2_ee[2] );
        temp = ( decoder2[1] - 1 ) & 0x0F;
        temp = Read_User_EE2( zone_name_address[temp], &decoder2_ee[0], 1 );// read zone name
        Line_2();                         // set lcd cursor to line2/position 1
        ptr = &room_name[decoder2_ee[0]][0];// pointer initialization
        printf("%s", ptr );               // format message for LCD
    }

    else
    {
            printf( "Error Condition " ); // format message for LCD
    }

    while ( valid_key != ESC );           // stay here until ESCape key is pressed
}

else if ( valid_key == BYPASS )           // BYPASS == 6 on keypad (labeled INSTANT)
{
    Home_Clr();                           // clear lcd and set cursor to line1/position 1
    printf("Lrn'd Xmtr's[1]");            // format message for LCD
    Line_2();                             // set lcd cursor to line2/position 1
    printf("Battery Stat[2]");            // format message for LCD

    key_wait = 0x00;                      // reset key wait timer
    while ( ( valid_key != ESC ) && ( key_wait < key_wait_limit ) )
    {
```

```
            if ( valid_key == '1' )
            {
                temp = Read_User_EE2( XMTR_CNT, &decoder2_ee[0], 1 );
                                        // read # of Xmtrs learned to EE
                Home_Clr();// clear lcd and set cursor to line1/position 1
                printf("Number of Xmtr's");   // format message for LCD
                Line_2();               // set lcd cursor to line2/position 1
                printf(" Learned-> %u", decoder2_ee[0]); // format message for LCD
                while ( valid_key !=  ESC );  // stay here until ESCape key is pressed
            }
            else if ( valid_key == '2' )
            {
                char index = 0x00;
                while ( valid_key != ESC )    // loop until ESCape key is pressed
                {
                    temp = Read_User_EE2( zone_name_address[index], &decoder2_ee[0], 5 ); // read
                    Home_It();              // set lcd cursor to line1/position 1
                    printf("Zone%u-> %02u:%02u:%02u", index+1, decoder2_ee[1], decoder2_ee[2],
                                            decoder2_ee[3] );
                    Line_2();               // set lcd cursor to line2/position 1
                    if ( decoder2_ee[4] == 0xAA )// test if battery state is Okay
                    {
                      printf("Battery Vdc:Okay" ); // format message for LCD
                    }
                    if ( decoder2_ee[4] == 0x55 )// test if battery state is Low
                    {
                      printf("Battery Vdc:Low " );// format message for LCD
                    }
                    else                    // else batteyr state not reported
                    {
                      printf("Battery Vdc:??? " ); // format message for LCD
                    }
                    index ++;               // increment zone name index
                    if ( index > 6 )        // test for more than 7 (0-6) zones
                    {
                      index = 0x00;         // increment zone name index
                    }
                    Delay_100mS( 1 );       // short delay
                    valid_key = 0x20;       // reset valid_key contents
                    while ( ( valid_key !=  '#' )  &&  ( valid_key !=  ESC ) );
                }

                temp = Read_User_EE2( LAST_XMIT, &decoder2_ee[0], 1 );
                Home_Clr();                 // clear lcd and set cursor to line1/position 1
                printf(" Last Reception " );  // format message for LCD
                Line_2();                   // set lcd cursor to line2/position 1
                if ( decoder2_ee[0] == ALRM )// test for Alarm transmission
                {
                    printf(" was an Alarm!  " ); // format message for LCD
                }
                if ( decoder2_ee[0] == CLEAR )// test for non-Alarm tranmsission
                {
                    printf(" was a non-Alarm" ); // format message for LCD
                }
                if ( decoder2_ee[0] == TST_LRN )// test for Learn/Test tranmsission
                {
                    printf("was a Test/Learn" );// format message for LCD
                }
                if ( decoder2_ee[0] == OKAY ) // test for 1.5 hour cycle transmission
                {
                    printf("was a Sensor OK " ); // format message for LCD
                }
                valid_key = 0x20;           // reset valid key contents
                while ( valid_key !=  ESC  ); // stay here until ESCape key is pressed
            }
        }
    }
}
Home_Clr();                             // clear lcd and set cursor to line1/position 1
break;


case ( AUX ):                           // if 'AUXiliary' key has been pressed

key_wait = 0x00;                        // reset key wait timer
while ( ( valid_key != ESC ) && ( key_wait < key_wait_limit ) )
{
       if ( valid_key == CHIME )  // CHIME == 9 on panel
       {
            Home_Clr();                 // clear lcd and set cursor to line1/position 1
            printf( "Enter key wait.." );     // format message for LCD
            Line_2();                   // set lcd cursor to line2/position 1
            printf( "[1]2Sec  [2]4ec " );     // format message for LCD
```

```
            key_wait = 0x00;                    // reset key wait timer
            while ( ( valid_key != ESC ) && ( key_wait < key_wait_limit ) )
            {
                switch ( valid_key )
                {
                    case ( '1' ):
                      key_wait_limit = SEC2;
                      key_wait = SEC4 + 1;      // set key wait timer to expire
                      break;
                    case ( '2' ):
                      key_wait_limit = SEC4;
                      key_wait = SEC4 + 1;      // set key wait timer to expire
                      break;
                      default:
                        break;
                }
            }
            Home_Clr();// clear lcd and set cursor to line1/position 1
            }

            else if ( valid_key == CODE )       // CODE == 8 on keypad
            {
                codestring[4] = 0x00;           // initialize variable
                csindex = 0;                    // define and initialize auto variable
                flag1.keyread = 0;              // reset key read flag

                Home_Clr();                     // clear lcd and set cursor to line1/position 1
                printf( " Enter 4-digit  " ); // format message for LCD
                Line_2();                       // set lcd cursor to line2/position 1
                printf( " Master Code..  " ); // format message for LCD

                Delay_100mS( 11 ) ;             // short delay for user to view LCD message
                Home_Clr();                     // clear lcd and set cursor to line1/position 1
                printf( "Mstr Code-> " );       // format message for LCD

                Code_Select();                  // function for entering Master/User code
                codestring[0] &= 0x0F;          //
                codestring[1] &= 0x0F;          //
                codestring[2] &= 0x0F;          //
                codestring[3] &= 0x0F;          //

/* At this point the 4-digit Master code may have been entered? */

                Home_Clr();                     // clear lcd and set cursor to line1/position 1
                if ( ( i = strcmp( codestring, master_code ) ) < 0 )
                {
                    flag1.code_valid = 0;       // flag set false, invalid master code entered
                }
                else if ( i > 0 )
                {
                    flag1.code_valid = 0;       // flag set false, invalid master code entered
                }
                else
                {
                    printf( "Valid Mastr Code" ); // format message for LCD
                    flag1.code_valid = 1;       // flag true if valid master code entered
                    Sound_Piezo( 1 );           // 100ms enable of internal piezo
                    Delay_100mS( 1 );           // short delay
                    Sound_Piezo( 1 );           // 100ms enable of internal piezo
                }

                if ( !flag1.code_valid )        // test if there was a Invalid Master Code entry
                {
                    printf( "* Wrong Master *" ); // format message for LCD
                    Line_2();                   // set lcd cursor to line2/position 1
                    printf( "* Code Entered *" ); // format message for LCD
                    Sound_Piezo( 1 );           // 100ms enable of internal piezo
                    Delay_100mS( 10 ) ;         // short delay for user to view LCD message
                    valid_key = ESC;            // ste valid_key for EScape key
                }

                else if ( flag1.code_valid )  // else test if there was a Valid Master Code entry
                {
                    Delay_100mS( 11 ) ;         // short delay for user to view LCD message
                    codestring[4] = 0x00;       // initialize variable
                    csindex = 0;                // define and initialize auto variable
                    valid_key = '#';            // set key entry point
                    flag1.keyread = 0;          // reset key read flag
                }

                while ( valid_key != ESC )      // stay in loop until ESCape is detected
                {
```

```
            if ( valid_key == '#' )    // test for # key detection
            {
              Home_Clr();              // clear lcd and set cursor to line1/position 1
              printf( " Enter 4-digit  " ); // format message for LCD
              Line_2();                // set lcd cursor to line2/position 1
              printf( " User Code..    " ); // format message for LCD

              Delay_100mS( 11 ) ;      // short delay for user to view LCD message
              Home_Clr();              // clear lcd and set cursor to line1/position 1
              printf( "User Code-> " );// format message for LCD

              Code_Select();           // function for entering Master/User code

              valid_key = 0x20;        // reset valid_key

              Home_Clr();              // clear lcd and set cursor to line1/position 1
              printf( "Accept Code:%s", &codestring[0] ); // format message for LCD
              Line_2();                // set lcd cursor to line2/position 1
              printf( "Yes[1] or No[2] " ); // format message for LCD

              while ( ( valid_key != ESC )  &&  ( valid_key != '#' ) )
              {
                if ( valid_key == '1' )// test if key 1 is pressed
                {
                  Home_Clr();          // clear lcd and set cursor to line1/position 1
                  printf( "  Code entered  " );// format message for LCD
                  Line_2();            // position lcd cursor on line2 / position 1
                  printf( "   Accepted!!   " );// format menu selection for LCD
                  temp = Write_User_EE2( USER_CODE, &codestring[0], 4 );// write to EE
                  Delay_100mS( 10 ) ;  // short delay for user to view LCD message
                  valid_key = ESC;
                }

                else if ( valid_key == '2' )
                {
                  Home_Clr();          // clear lcd and set cursor to line1/position 1
                  printf( "  Code entered  " );// format message for LCD
                  Line_2();            // position lcd cursor on line2 / position 1
                  printf( "  not Accepted! " );// format message for LCD
                  csindex = 0x00;      // reset code array index
                  flag1.keyread = 0;   // reset key read flag
                  Delay_100mS( 7 );    // short delay for user to view LCD message
                  valid_key = '#';     // set valid_key to stay in loop
                }
              }
            }
        }
        Home_Clr();                    // clear lcd and set cursor to line1/position 1
    }

    else if ( valid_key == TIME )        // TIME == 7 on keypad
    {
        Home_Clr();                    // clear lcd and set cursor to line1/position 1
        printf( " Toggle Time[1] " ); // format message for LCD
        Line_2();                      // set lcd cursor to line2/position 1
        printf( " or Set Time[2] " ); // format message for LCD

        key_wait = 0x00;               // reset key wait timer
        while ( ( valid_key != ESC ) && ( key_wait < key_wait_limit ) )
        {
            if ( valid_key ==  DISPLAY_TIME ) // DISPLAY_TIME == 1 on keypad
            {
              flag1.time_update ^= 1;  // toggle time on/off
              Home_Clr();              // clear lcd and set cursor to line1/position 1

              if ( flag1.time_update != 1) // LCD toggled off
              {
                Line_2();              // set lcd cursor to line2/position 1
                printf( "      " );    // format message for LCD
              }
              key_wait = SEC4 + 1;     // set key wait timer to expire
            }

            else if ( valid_key == SET_TIME ) // SET_TIME == 2 on keypad
            {
              flag1.keyread = 0;       // reset key read flag
              Set_Time();              // function to set time
              key_wait = SEC4 + 1;     // set key wait timer to expire
            }
        }
    }

    else if ( valid_key == LEARN )    // LEARN == 6 on keypad (labled BYPASS on panel)
```

```
    {
        Home_Clr();                  // clear lcd and set cursor to line1/position 1
        printf( "  Learn Mode[1] " ); // format message for LCD
        Line_2();                    // set lcd cursor to line2/position 1
        printf( "or Erase All[2] " ); // format message for LCD

        key_wait = 0x00;             // reset key wait timer
        while ( ( valid_key != ESC ) && ( key_wait < key_wait_limit ) )
        {
            if ( valid_key == LEARN_DEC )// LEARN_DEC == 1 on keypad
            {
              Home_Clr();            // clear lcd and set cursor to line1/position 1
              temp = Learn();        // execute decoder learn function
              if ( temp == 0 )       // chekc for no error during learn
              {
                printf(" Learn Entered! " ); // format message for LCD
                Line_2();            // set lcd cursor to line2/position 1
                printf(" 35sec. timeout " ); // format message for LCD
                Delay_100mS( 4 );    // short delay
                Sound_Piezo( 1 );    // quick toggle of internal piezo
              }
              else
              {
                printf("Lrn Entry Error!" ); // format message for LCD
              }
              key_wait = SEC4 + 1;   // set key wait timer to expire
            }

            else if ( valid_key == ERASE )   // ERASE == 2 on keypad
            {
              Home_Clr();            // clear lcd and set cursor to line1/position 1
              temp = Erase_All();    // execute decoder erase all function
              if ( temp == 0 )       // check for no error during erase all
              {
                printf("   Erase All    " ); // format message for LCD
                Line_2();            // set lcd cursor to line2/position 1
                printf("  Successful!   " ); // format message for LCD
                Delay_100mS( 4 );    // short delay
                Sound_Piezo( 1 );    // quick toggle of internal piezo
                decoder2_ee[0] = 0x00;
                temp = Write_User_EE2( XMTR_CNT,  &decoder2_ee[0], 1 );
              }
              else
              {
                printf("Erase Entry Err " ); // format message for LCD
              }
              Home_Clr();            // clear lcd and set cursor to line1/position 1
              key_wait = SEC4 + 1;   // set key wait timer to expire
            }
        }
    }
}
break;


case ( ESC ):

    key_wait = 0x00;                 // reset key wait timer
    valid_key = 0x00;                // reset valid key contents
    while ( ( valid_key != ESC ) && ( key_wait < key_wait_limit ) )
    {
        if ( valid_key ==  '#' )     // labled '#' on keypad
        {
            Home_Clr();              // clear lcd and set cursor to line1/position 1
            if ( ( !flag2.alarm_set1 )  &&  ( !flag2.alarm_set2 )  &&
                                     //  ( !flag1.arm_countdown ) )
            {
              printf( "System not Armed" ); // format message for LCD
              Delay_100mS( 10 ) ;    // short delay for user to view LCD message
              Home_Clr();            // clear lcd and set cursor to line1/position 1
              valid_key = ESC;       // set valid_key contents to ESCape
            }

            else
            {
              codestring[4] = 0x00;  // initialize variable
              csindex = 0;           // define and initialize auto variable
              flag1.keyread = 0;     // reset key read flag

              printf( "To Disarm..Enter" ); // format message for LCD
              Line_2();              // set lcd cursor to line2/position 1
              printf( "User Code-> " );// format message for LCD
```

```
                    Code_Select();          // function for entering Master/User code

                    temp = Read_User_EE2( USER_CODE, &tempstring[0], 4 ); // read from to EE

                    Home_Clr();              // clear lcd and set cursor to line1/position 1
                    if ( ( i = strcmp( codestring, tempstring ) ) < 0 )
                    {
                      flag1.code_valid = 0;  // flag set false, invalid master code entered
                    }
                    else if ( i > 0 )
                    {
                      flag1.code_valid = 0;  // flag set false, invalid master code entered
                    }
                    else
                    {
                      Home_It();            // set lcd cursor to line1/position 1
                      printf("System DisArmed!"); // format message for LCD
                      ALARM_OFF;            // turn off external Alarm drive
                      ARMRDY = 1;           // turn on base panel ARMED LED
                      Sound_Piezo( 1 );     // 100ms enable of internal piezo
                      flag1.code_valid = 1; // flag set false, invalid master code entered
                      flag2.alarm_set1= 0;  // reset alarm state entry flag1
                      flag2.alarm_set2= 0;  // reset alarm state entry flag2
                      flag1.arm_now = 0;    // reset immediate alarm entry flag
                      flag1.arm_countdown = 0;// ensure flag is reset
                    }

                    if ( !flag1.code_valid )// test if there was a Invalid Master Code entry
                    {
                      printf( "* Invalid Code *" ); // format message for LCD
                    }

                    Delay_100mS( 10 ) ;     // short delay for user to view LCD message
                    Home_Clr();             // clear lcd and set cursor to line1/position 1
                    valid_key = ESC;
                  }
              }

              else if ( valid_key == '0' )  //
              {
                  Home_Clr();// clear lcd and set cursor to line1/position 1
                  valid_key = ESC;// set valid key contents to ESCape
              }


              else if ( valid_key ==  '1' ) // labled 1 on keypad
              {
                  flag1.buzzer = 0;// clear flag so piezo buzzer will not sound
                  valid_key = ESC;// set valid key contents to ESCape
              }

              else if ( valid_key ==  PANIC )//
              {
                  decoder2_ee[0] = CLEAR;   // write non-alarm status byte to buffer
                  temp = Write_User_EE2( ALRM_STAT, &decoder2_ee[0], 1 ); // write alarm time to
                                            // EE memory
              }

              else if ( valid_key ==  ESC )//
              {
                  if  ( !flag1.battery_sel )
                  {
                    BATSEL ^= 1;            // toggle battery backup select
                  }

                  LCDBD ^= 1;               // toggle LCD backlight on/off
              }
          }
        break;

        default:
        break;
    }
}
```

```
/************************************************************************
*                                                                       *
*   Filename:       prockey.h                                           *
*   Date:           07/18/99                                            *
*   File Version:   1.00                                                *
*                                                                       *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                    *
*                                                                       *
************************************************************************/



// FUNCTION PROTOTYPES

/* Functions defined in file baselcd.c */
extern void Home_Clr( void );               // reference linkage to defined function
extern void Home_It( void );
extern void Line_2( void );

/* Functions defined in file diagfunc.c */
extern void Sound_Piezo( char ontime );     // reference linkage to defined function
extern void Battery_Voltage( void );

/* Functions defined in file delays.c */
extern void Delay_1S( char loop_count );    // reference linkage to defined function
extern void Delay_100mS( char loop_count );
extern void Delay_1mS( char loop_count );

/* Functions defined in file hcsdec.c */
extern char Learn( void );                  // reference linkage to defined function
extern char Erase_All( void );
extern char Read_User_EE2( char address, char * rdptr, char length );
extern char Write_User_EE2( char address, char * wrptr, char length );

/* Function defined in file timeset.c */
extern void Set_Time( void );               // reference linkage to defined function

/* Function defined in file codesel.c */    // reference linkage to defined function
extern void Code_Select( void );


// VARIABLES ( REFERENCE DECLARATION )

extern struct event_bits1                   // bit structure for housekeeping flags
      {
        unsigned new_day         :1;        // flag for indicating new day
        unsigned arm_countdown   :1;        // flag set when counting down to arming system
        unsigned buzzer          :1;        // flag set when piezo buzzer will sound when called
        unsigned read_battery    :1;        // flag used for read battery voltage
        unsigned battery_on      :1;        // flag set when battery source is selected
        unsigned battery_off     :1;        // flag set when battery is not selected
        unsigned time_update     :1;        // flag used for updating LCD and E2 Memory
        unsigned erase_all       :1;        // flag set when HCS515 erase all operation complete
        unsigned battery_sel     :1;        // flag set when battery source is selected
        unsigned erase_entered   :1;        // flag set when HCS515 erase all operation entered
        unsigned arm_now         :1;        // flag set when system is immediately armed w/o user
                                            // code
        unsigned learn_entered   :1;        // flag set when HCS515 learn operation entered
        unsigned code_valid      :1;        // flag set after user security access code accepted
        unsigned code_entered    :1;        // flag set when user security access code entered
        unsigned keyread         :1;        // flag set for indicating keypad selection needs
                                            // processed
        unsigned keyhit          :1;        // flag set when valid key hit on 4x4 keypad is
                                            // detected
    } flag1;                                // variable name


extern struct event_bits2                   // define bit structure for housekeeping flags
    {
        unsigned                 :1;        // bit padding
        unsigned alarm_set1      :1;        // flag set when system is armed ( 1 of 2 )
        unsigned                 :6;        // bit padding
        unsigned valid_rcv       :1;        // flag used if HCS515 demod data is correct
        unsigned sensor_batt_low :1;        // flag indicating if sensor module battery is low
        unsigned                 :5;        // bit padding
        unsigned alarm_set2      :1;        // flag set when system is armed ( 2 of 2 )
    } flag2;                                // variable name


extern char porta_image;                    // reference linkage to defined variable in bank0
extern char hours;                          // reference linkage to defined variable in bank0
extern char minutes;                        // reference linkage to defined variable in bank0
```

```
extern char seconds;                            // reference linkage to defined variable in bank0
extern char key_index;                          // reference linkage to defined variable in bank0
extern bank1 char key_wait;                     // reference linkage to defined variable in bank1
extern bank1 char valid_key;                    // reference linkage to defined variable in bank1
extern bank1 char key_wait_limit;               // reference linkage to defined variable in bank1
extern bank1 char temp;                         // reference linkage to defined variable in bank1

extern bank1 char decoder2[10];                 // reference linkage, array storage for valid trans.
                                                // reception (decoder 2)
extern bank1 char decoder2_ee[6];               // reference linkage to defined variable in bank1
extern bank1 double battery_volts;              // reference linkage to defined variable in bank2


extern const char zone_tod_address[];
extern const char zone_name_address[];
extern const char room_name[][17];

extern const char *ptr;
extern const char master_code[];                // Master code required for changing user code


// VARIABLES ( DEFINED HERE )

bank1 char codestring[5];                       // define array variable in bank1
bank1 char csindex;                             // define array index variable in bank1
bank1 char tempstring[5];                       // define array variable for temp storage
bank1 char time_to_arm;                         // define variable


// PORTBITS ( DEFINED HERE )

#define PortBit(port,bit)    ((unsigned)&(port)*8+(bit))

static bit ALARM_DRV  @  PortBit(PORTA,2);      // Alarm drive state
static bit ARMRDY     @  PortBit(PORTC,3);      // Alarm/Ready Light bias control
static bit PSOURCE    @  PortBit(PORTD,0);      // Power Source indication status
static bit BATSEL     @  PortBit(PORTE,0);      // Battery Select (test only)
static bit LCDBD      @  PortBit(PORTE,2);      // LCD Back Drive on/off


// MACROS ( DEFINED HERE )

#define NOP() asm(" nop")// define NOP macro

#define ALARM_ON     porta_image |= 0b000100;\
                     PORTA = porta_image;       // enable external alarm

#define ALARM_OFF    porta_image &= ~0b000100;\
                     PORTA = porta_image;       // enable external alarm

#define BATT_CURR_ON porta_image |= 0b000010 ;\
                     PORTA = porta_image;       // enable maximum battery charge current

#define BATT_CURR_OFFporta_image &= ~0b000010 ;\
                     PORTA = porta_image;       // enable maximum battery charge current


#define    SEC2          61                     //
#define    SEC4          120                    //

#define    ESC           0x1B                   // text sub. for Escape key
#define    AWAY          0x31                   // text sub. for '1' key
#define    YES           0x31                   // text sub. for '1' key
#define    DISPLAY_TIME  0x31                   // text sub. for '1' key
#define    LEARN_DEC     0x31                   // text sub. for '1' key

#define    OFF           0x32                   // text sub. for '2' key
#define    SET_TIME      0x32                   // text sub. for '2' key
#define    ERASE         0x32                   // text sub. for '1' key

#define    TEST          0x35                   // text sub. for '5' key
#define    LEARN         0x36                   // text sub. for '6' key
#define    BYPASS        0x36                   // text sub, for '6' key
#define    TIME          0x37                   // text sub. for '7' key
#define    ALARM_STATUS  0x37                   // text sub. for '7' key
#define    CODE          0x38                   // text sub. for '8' key
#define    CHIME         0x39                   // text sub. for '9' key

#define    ALT           0x40                   // text sub. for ALTernate key
#define    AUX           0x41                   // text sub. for Auxillary key
#define    PANIC         0x50                   // text sub. for PANIC key
```

```
/*****************************************************************
*                                                                *
*   Wireless Home Security with Keeloq and the PICmicro          *
*                                                                *
*****************************************************************
*                                                                *
*   Filename:       proctran.c                                   *
*   Date:           07/18/99                                     *
*   File Version:   1.00                                         *
*                                                                *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3             *
*                                                                *
*   Author:         Richard L. Fischer                           *
*   Company:        Microchip Technology Incorporated            *
*                                                                *
*****************************************************************
*                                                                *
*   Files required:                                              *
*                                                                *
*                   pic.h                                        *
*                   proctran.h                                   *
*                   hcs515ee.h                                   *
*                   errcode.h                                    *
*                                                                *
*****************************************************************
*                                                                *
*   Notes: The routines within this file are required for        *
*          determining what type of transmission has been received *
*          from the HCS515 decoder(s).                           *
*                                                                *
*   Currently only 1 decoder is implemented.                     *
*                                                                *
*****************************************************************/

    #include <pic.h>
    #include "proctran.h"                    // function prototypes, defines..
    #include "hcs515ee.h"

bit alarm_detect;                            // define bit
bit ok_detect;                               // define bit
bit test_detect;                             // define bit


void Read_Decoder_Trans( void )
{
    char maj_detect = 0;                     // initialize variable for majority detect

    alarm_detect = 0;                        // initialize bit
    ok_detect = 0;                           // initialize bit
    test_detect = 0;                         // initialize bit
    flag2.valid_rcv = 0;                     // reset valid reception flag

    Delay_1mS( 2 );                          // wait 2mS
    if ( HCSDATA2 )                          // is data line high?
        maj_detect ++;                       // yes so increment majority detect count

    Delay_1mS( 2 );                          // wait 2mS
    if ( HCSDATA2 )                          // is data line high?
        maj_detect ++;                       // yes so increment majority detect count

    Delay_1mS( 20 );                         // wait 20mS

    if ( HCSDATA2 )                          // is data line high?
        maj_detect ++;                       // yes so increment majority detect count

    Delay_1mS( 2 );                          // wait 2mS
    if ( HCSDATA2 )                          // is data line high?
        maj_detect ++;                       // yes so increment majority detect count

    if ( maj_detect == 4 )                   // majority detect == 4?
    {
        Read_Trans( 10 );                    // can read up to 10 bytes of received message

/* Test here if received function codes indicate a possible "ALARM" transmission */
        if (  (( decoder2[0] & 0x78 ) == 0x08 )  &&  (( decoder2[5] & 0xF0 ) == 0x20 )  )
        {
            alarm_detect = 1;
            flag2.valid_rcv = 1;             // if here then there is a valid reception
        }

/* Test here if received function codes indicate a "TEST/LEARN" transmission */
        else if (  (( decoder2[0] & 0x78 ) == 0x10 )  &&  (( decoder2[5] & 0xF0 ) == 0x40 )  )
        {
```

```
            test_detect = 1;
            flag2.valid_rcv = 1;              // if here then there is a valid reception
        }

/* Test here if received function codes indicate a possible "OKAY" transmission */
        else if (  (( decoder2[0] & 0x78 ) == 0x18 )  &&  (( decoder2[5] & 0xF0 ) == 0x60 )  )
        {
            ok_detect = 1;
            flag2.valid_rcv = 1;              // if here then there is a valid reception
        }
        maj_detect = 0;                       // reset majority detect count
    }

    if ( flag2.valid_rcv )                    // test if there was a valid transmission
    {
        decoder2_ee[0] = hours;               // write hours count to buffer for EE write
        decoder2_ee[1] = minutes;             // write minutes count to buffer for EE write
        decoder2_ee[2] = seconds;             // write seconds count to buffer for EE write

        if ( alarm_detect )
        {
            if ( ( flag2.alarm_set1 == 1 )  &&  ( flag2.alarm_set2 == 1 ) ) // is system ARMED?
            {
                ALARM_ON;                     // yes, so turn on external alarm
                Sound_Piezo( 5 );             // toggle internal piezo for 500mS
                Delay_100mS( 1 );             // short delay
                Sound_Piezo( 1 );             // toggle internal piezo for 100mS
                decoder2_ee[3] = ALRM;        // write "ALARM ON" status byte to buffer for EE
write
            }
            else                              // if here zone trip occurred but system is NOT ARMED
            {
                ALARM_OFF;                    // ensure alarm is off
                Sound_Piezo( 1 );             // toggle internal piezo for 100mS
                Delay_100mS( 1 );             // short delay
                Sound_Piezo( 5 );             // toggle internal piezo for 500mS
                decoder2_ee[3] = CLEAR;       // write "CLEAR" status byte to buffer for EE write
            }
            temp = Write_User_EE2( ALRM_HRS, &decoder2_ee[0], 4 ); // write alarm condition status
                                              // and time to EE
        }

        else if ( test_detect )
        {
            Sound_Piezo( 1 );// toggle internal piezo for 100mS
            decoder2_ee[3] = TST_LRN;         // write "TEST/LEARN" status byte to buffer for EE
                                              // write
        }


        else if ( ok_detect )
        {
            Sound_Piezo( 1 );                 // toggle internal piezo for 100mS
            Delay_100mS( 1 );                 // short delay
            Sound_Piezo( 1 );                 // toggle internal piezo for 100mS
            decoder2_ee[3] = OKAY;            // write "OKAY" status byte to buffer for EE write
        }

        temp = Write_User_EE2( LAST_XMIT, &decoder2_ee[3], 1 ); // write last transmission type
                                              // code to EE

        if ( ( decoder2[0] & 0x04 ) )         // test for Vlow bit in sensor module transmission
        {
            decoder2_ee[3] = LOW;             // write battery "LOW" status code to buffer for EE
                                              // write
            flag2.sensor_batt_low = 1;        // set flag indicating zone battery is low
        }
        else
        {
            decoder2_ee[3] = HIGH;            // write battery "HIGH" status code to buffer for EE
                                              // write
            flag2.sensor_batt_low = 0;        // reset flag
        }

/* Write battery status and respective time stamp to appropriate zone tag in EE memory */
            temp = ( decoder2[1] - 1) & 0x0F;// determine which transmitter block was received
            temp = Write_User_EE2( zone_tod_address[temp], &decoder2_ee[0], 4 ); // write alarm
                                              // time to EE memory
    }
}
```

```
/*********************************************************************
*                                                                    *
*   Filename:       proctran.h                                       *
*   Date:           07/18/99                                         *
*   File Version:   1.00                                             *
*                                                                    *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                 *
*                                                                    *
*********************************************************************/


// FUNCTION PROTOTYPES

/* Functions defined in file delays.c */
extern void Delay_100mS( char loop_count );  // refercene linkage to defined function
extern void Delay_10mS( char loop_count );
extern void Delay_1mS( char loop_count );

/* Function defined is file diagfunc.c */
extern void Sound_Piezo( char ontime );      // reference linkage to defined function

/* Functions defined in file hcsdec.c */
extern void Read_Trans( char length );       // reference linkage to defined function
extern char Write_User_EE2( char address, char * wrptr, char length );
extern char Read_User_EE2( char address, char * rdptr, char length );


const char zone_tod_address[] = { 0x8A, 0x91, 0x98, 0x9F, 0xA6, 0xAD, 0xB4 };


// VARIABLES ( REFERENCE DECLARATION )

extern struct event_bits2                    // define bit structure for housekeeping flags
    {
        unsigned                :1;          // bit padding
        unsigned alarm_set1     :1;          // flag set when system is armed ( 1 of 2 )
        unsigned                :6;          // bit padding
        unsigned valid_rcv      :1;          // flag used if HCS515 demod data is correct
        unsigned sensor_batt_low :1;         // flag indicating if sensor module battery is low
        unsigned                :5;          // bit padding
        unsigned alarm_set2     :1;          // flag set when system is armed ( 2 of 2 )
    } flag2;                                 // variable name


extern char porta_image;                     // reference linkage to defined variable (SFR)
extern char hours;                           // reference linkage to defined variable in bank0
extern char minutes;                         // reference linkage to defined variable in bank0
extern char seconds;                         // reference linkage to defined variable in bank0


extern bank1 char decoder2[10];              // reference linkage, array storage for valid trans.
                                             // reception (decoder 2)
extern bank1 char decoder1_ee[6];            // reference linkage to defined array variable in
                                             // bank1
                                             // reception (decoder 2)
extern bank1 char decoder2_ee[6];            // reference linkage to defined array variable in
                                             // bank1

extern bank1 char temp;                      // reference linkage to defined variable in bank1


// PORTBITS ( DEFINED HERE )

#define PortBit(port,bit)    ((unsigned)&(port)*8+(bit))

static bit HCSDATA2   @  PortBit(PORTC,5);   // declare bit for HCS515 data line


// MACROS ( DEFINED HERE )

#define NOP()asm(" nop")                     // define NOP macro

#define ALARM_ON       porta_image |= 0b000100;\
                       PORTA = porta_image;  // enable external alarm

#define ALARM_OFF      porta_image &= ~0b000100;\
                       PORTA = porta_image;  // enable external alarm
```



```
/************************************************************************
*                                                                       *
*   Wireless Home Security with Keeloq and the PICmicro                 *
*                                                                       *
************************************************************************
*                                                                       *
*   Filename:       timeset.c                                           *
*   Date:           07/18/99                                            *
*   File Version:   1.00                                                *
*                                                                       *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                    *
*                                                                       *
*   Author:         Richard L. Fischer                                  *
*   Company:        Microchip Technology Incorporated                   *
*                                                                       *
************************************************************************
*                                                                       *
*   Files required:                                                     *
*                                                                       *
*                   pic.h                                               *
*                   stdio.h                                             *
*                   time.h                                              *
*                                                                       *
************************************************************************
*                                                                       *
*   Notes: The routine within this file is used for                     *
*          setting the time-of-day via 4x4 keypad selections.           *
*                                                                       *
*   Key         Function                                                *
*                                                                       *
*    1          Increment hours count                                   *
*    4          Decrement hours count                                   *
*                                                                       *
*    2          Increment minutes count                                 *
*    5          Decrement minutes count                                 *
*                                                                       *
*    3          Increment seconds count                                 *
*    6          Decrement seconds count                                 *
*                                                                       *
*                                                                       *
************************************************************************/

    #include <pic.h>                        // processor if/def file
    #include <stdio.h>
    #include "time.h"                       // function prototypes, variables, defines..


void Set_Time( void )
{
    TMR1IE = 0;                             // disable real-time clock interrupts
    Home_Clr();                             // clear lcd and set cursor to line1/position 1

    while ( valid_key != ESC )
    {
        Line_2();                           // set lcd cursor to line2/position 1
        Delay_10mS( 15 );                   // short delay
        printf( "Time-> %02u:%02u:%02u" ,hours,minutes,seconds );

        switch ( valid_key )                // evaluate expression
        {
            case ( '1' ):                   // test if hours will be incremented
                hours++;                    // increment hours
                if ( hours > 23 )           // test if hours will roll over
                {
                    hours = 0x00;           // yes, so reset hours
                }
                break;                      // exit from switch evaluation

            case ( '2' ):                   // test if minutes will be incremented
                minutes++;                  // increment minutes
                if ( minutes > 59  )        // test if minutes will roll over
                {
                    minutes = 0x00;         // yes, so reset minutes
                }
                break;                      // exit from switch evaluation

            case ( '3' ):                   // test if seconds will be incremented
                seconds++;                  // increment seconds
                if ( seconds > 59 )         // test if seconds will roll over
                {
                    seconds = 0x00;         // yes, so reset seconds
                }
```

```
                break;                          // exit from switch evaluation

            case ( '4' ):                       // test if hours will be decremented
                hours--;                        // decrement hours
                if ( hours > 23 )               // test if hours will underflow
                {
                     hours = 23;                // yes, so reset hours
                }
                break;                          // exit from switch evaluation

            case ( '5' ):                       // test if minutes will be decremented
                minutes--;                      // decrement minutes
                if ( minutes > 59 )             // test if minutes will underflow
                {
                     minutes = 59;              // yes, so reset minutes
                }
                break;                          // exit from switch evaluation

            case ( '6' ):                       // test if seconds will be decremented
                seconds--;                      // decrement seconds
                if ( seconds > 59 )             // test if seconds will underflow
                {
                seconds = 59;                   // yes, so reset seconds
                }
                break;                          // exit from switch evaluation

            default:                            // if no match occurs
                break;                          // exit from switch evaluation
        }
    }

    TMR1IF = 0;                                 // reset Timer 1 interrupt flag
    TMR1IE = 1;                                 // re-enable Timer 1 interrupts
    Home_It();                                  // set lcd cursor to line1/position 1
    printf( " Time Entered!  ");                // format message for display
    Delay_100mS( 10 );                          // short delay
    Home_Clr();                                 // clear lcd and set cursor to line1/position 1
    flag1.time_update = 1;                      // set flag for displaying time
}
```

```
/*********************************************************************
*                                                                    *
*   Filename:       time.h                                           *
*   Date:           07/18/99                                         *
*   File Version:   1.00                                             *
*                                                                    *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                 *
*                                                                    *
*********************************************************************/



// FUNCTION PROTOTYPES

/* Functions defined in file baselcd.c */
extern void Home_Clr(void);                  // reference linkage to defined function
extern void Home_It(void);
extern void Line_2(void);

/* Functions defined in file delays.c */
extern void Delay_100mS( char loop_count );  // reference linkage to defined function
extern void Delay_10mS( char loop_count );


// VARIABLES ( REFERENCE DECLARATION )

extern struct event_bits1                    // bit structure for housekeeping flags
      {
        unsigned new_day         :1;         // flag for indicating new day
        unsigned arm_countdown   :1;         // flag set when counting down to arming system
        unsigned buzzer          :1;         // flag set when piezo buzzer will sound when called
        unsigned read_battery    :1;         // flag used for read battery voltage
        unsigned battery_on      :1;         // flag set when battery source is selected
        unsigned battery_off     :1;         // flag set when battery is not selected
        unsigned time_update     :1;         // flag used for updating LCD and E2 Memory
        unsigned erase_all       :1;         // flag set when HCS515 erase all operation complete
        unsigned battery_sel     :1;         // flag set when battery source is selected
        unsigned erase_entered   :1;         // flag set when HCS515 erase all operation entered
        unsigned arm_now         :1;         // flag set when system is immediately armed w/o user
                                             // code
        unsigned learn_entered   :1;         // flag set when HCS515 learn operation entered
        unsigned code_valid      :1;         // flag set after user security access code accepted
        unsigned code_entered    :1;         // flag set when user security access code entered
        unsigned keyread         :1;         // flag set for indicating keypad selection needs
                                             // processed
        unsigned keyhit          :1;         // flag set when valid key hit on 4x4 keypad is
                                             // detected
    } flag1;                                 // variable name

extern char hours;                           // reference linkage to defined variable in bank0
extern char minutes;                         // reference linkage to defined variable in bank0
extern char seconds;                         // reference linkage to defined variable in bank0
extern bank1 char valid_key;                 // reference linkage to defined variable in bank1



// MACROS DEFINED HERE

#define NOP()  asm(" nop")                   // define NOP macro

#define   SEC4      120                      //
#define   ESC      0x1B                      // text sub. for Escape key
```

```
/*****************************************************************
*                                                                *
*   Wireless Home Security with Keeloq and the PICmicro          *
*                                                                *
*****************************************************************
*                                                                *
*   Filename:       zonename.c                                   *
*   Date:           07/18/99                                     *
*   File Version:   1.00                                         *
*                                                                *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3              *
*                                                                *
*   Author:         Richard L. Fischer                           *
*   Company:        Microchip Technology Incorporated            *
*                                                                *
*****************************************************************
*                                                                *
*   Files required:                                              *
*                                                                *
*                   pic.h                                        *
*                   stdio.h                                      *
*                   zonename.h                                   *
*                   hcs515ee.h                                   *
*                                                                *
*****************************************************************
*                                                                *
*   Notes: The routine within this file are used for             *
*          assigning key (room) names to learned sensor          *
*          modules (zones).                                      *
*                                                                *
*                                                                *
*****************************************************************/

    #include <pic.h>
    #include <stdio.h>
    #include "zonename.h"                   // function prototypes, defines..
    #include "hcs515ee.h"                   // HCS515 EE memory defines


void Zone_Name( void )
{
    char room_index = 0x00;                 // define and init auto variable
    T0IE = 0;                               // disable Timer0 interrupts

    Home_Clr();                             // clear lcd & set cursor to line1/position1
    printf( "Select Zone Name" );           // format message for LCD
    Delay_100mS( 15 );                      // ~1.5 second delay
    Home_Clr();                             // clear lcd & set cursor to line1/position1
    printf( "[ESC] == choice " );           // format message for LCD
    valid_key = 0x20;                       // reset valid key buffer contents

    while ( valid_key != ESC )              // loop until ESC key is detected
    {
        Line_2();                           // set cursor to line2 / position 1
        ptr = &room_name[room_index][0];    // pointer initialization to names for rooms
        printf("%s", ptr );                 // format message for LCD
        Delay_10mS( 8 );                    // short response delay

        if ( valid_key == '1' )             // are we scrolling through zone names?
        {
            room_index++;                   // increment room index count
            valid_key = 0x20;               // reset valid key entry
            if ( room_index > max_rooms )   // test if room index exceeds max # of rooms
            {
                room_index = 0x00;          // reset room index count
            }
        }
    }

    decoder2_ee[0] = ( ( decoder2[1]  >> 4 ) & 0x0F ); // determine number of Xmtrs learned
    temp = Write_User_EE2( XMTR_CNT, &decoder2_ee[0], 1 ); // write number Xmtrs learned to EE

    decoder2_ee[0] = room_index;            // acquire reference count for zone name
    temp = ( decoder2[1] - 1 ) & 0x0F;      // determine which transmitter block was just
                                            // received
    temp = Write_User_EE2( zone_name_address[temp], &decoder2_ee[0], 1 ); // write zone name to EE
                                            // memory

    Home_Clr();                             // clear lcd & set cursor to line1/ position1
    valid_key = 0x20;
    T0IE = 1;                               // re-enable Timer0 based interrupts
}
```

```
/*********************************************************************
*                                                                    *
*   Filename:       zonename.h                                       *
*   Date:           07/18/99                                         *
*   File Version:   1.00                                             *
*                                                                    *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                 *
*                                                                    *
*********************************************************************/



// FUNCTION PROTOTYPES

/* Functions defined in file baselcd.c */
extern void Home_Clr(void);                 // reference linkage to defined function
extern void Line_2(void);

/* Functions defined in file delays.c */
extern void Delay_10mS( char loop_count );  // reference linkage to defined function
extern void Delay_100mS( char loop_count );


// VARIABLES ( DEFINED HERE )

const char *ptr;                            // define general purpose const pointer

const char zone_name_address[] = { 0x89, 0x90, 0x97, 0x9E, 0xA5, 0xAC, 0xB3 };

const char room_name[][17] = { "  Family Room  ","  Living Room  ","  Great Room  ",
                               "  Dining Room  ","  Den  ","  Office  ",
                               "  Master Bedroom  ","  Bedroom #1  ","  Bedroom #2  ",
                               "  Bedroom #3  ","  Garage Door  ","  Service Door ",
                                "  Laundry Door  ","  Patio Door  ","  Sliding Door  " };
#define max_rooms  14



// VARIABLES ( REFERENCE DECLARATION )

extern struct event_bits2                   // define bit structure for housekeeping flags
    {
        unsigned                :1;         // bit padding
        unsigned alarm_set1     :1;         // flag set when system is armed ( 1 of 2 )
        unsigned                :6;         // bit padding
        unsigned valid_rcv      :1;         // flag used if HCS515 demod data is correct
        unsigned sensor_batt_low:1;         // flag indicating if sensor module battery is low
        unsigned                :5;         // bit padding
        unsigned alarm_set2     :1;         // flag set when system is armed ( 2 of 2 )
    } flag2;                                // variable name


extern bank1 char key_wait;                 // reference linkage to defined variable in bank1
extern bank1 char valid_key;                // reference linkage to defined variable in bank1
extern bank1 char temp;                     // reference linkage to defined variable in bank1

extern bank1 char decoder2_ee[6];
extern bank1 char decoder2[10];             // array storage for valid trans.

extern char Write_User_EE2( char address, char * wrptr, char length );



// MACROS ( DEFINED HERE )

#define NOP() asm(" nop")                   // define NOP macro

#define   ESC           0x1B                // text sub. for Escape key
```

```
;*****************************************************************
;                                                                *
;   Wireless Home Security with Keeloq and the PICmicro          *
;                                                                *
;*****************************************************************
;                                                                *
;   Filename:       powerup77.as                                 *
;   Date:           07/18/99                                     *
;   File Version:   1.00                                         *
;                                                                *
;*****************************************************************
;                                                                *
;   Notes: This assembly file provides for a remapped processor  *
;          code startup location.                                *
;                                                                *
;                                                                *
;   Within the  "#if defined(_PIC14)" section below the new startup *
;   code  called "mystartup" is executed upon successful completion *
;   of a reset state. The function "mystartup" is located in the *
;   base77.c file.                                               *
;                                                                *
;                                                                *
;*****************************************************************

    #include"sfr.h"


    global  powerup,start
    psect   powerup,class=CODE,delta=2

extrn  _mystartup

powerup
#if defined(_12C508) || defined(_12C509)
    movwf   5       ;store calibration to OSCCAL
#endif
#if defined(_PIC14)
    movlw   high _mystartup
    movwf   PCLATH
    goto    (_mystartup & 0x7FF)
#endif
#if defined(_PIC16)
    movlw   start>>8
    movwf   PCLATH
    movlw   start & 0xFF
    movwf   PCL
#endif
    end powerup
```

```
/************************************************************************
*                                                                       *
*   Filename:       cnfig77.h                                           *
*   Date:           07/18/99                                            *
*   File Version:   1.00                                                *
*                                                                       *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                    *
*                                                                       *
************************************************************************/

/*****  CONFIGURATION BIT DEFINITIONS FOR PIC16C77 PICmicro  *****/


#define    CONBLANK     0x3FFF

#define    CP_ALL       0x00CF
#define    CP_75        0x15DF
#define    CP_50        0x2AEF
#define    CP_OFF       0x3FFF
#define    BODEN_ON     0x3FFF
#define    BODEN_OFF    0x3FBF
#define    PWRTE_OFF    0x3FFF
#define    PWRTE_ON     0x3FF7
#define    WDT_ON       0x3FFF
#define    WDT_OFF      0x3FFB
#define    LP_OSC       0x3FFC
#define    XT_OSC       0x3FFD
#define    HS_OSC       0x3FFE
#define    RC_OSC       0x3FFF
```

```
;*****************************************************************************
;                                                                            *
;   Wireless Home Security with Keeloq and the PICmicro                      *
;                                                                            *
;*****************************************************************************
;                                                                            *
;   Filename:       basecode.as                                              *
;   Date:           07/18/99                                                 *
;   File Version:   1.00                                                     *
;                                                                            *
;*****************************************************************************
;                                                                            *
;   Notes: This assembly file provides for the Master Code                   *
;          which can be placed into EPROM at programming time.               *
;                                                                            *
;   The address location in EPROM for this PSECT ( accesscode )              *
;   is 1FFAh.                                                                *
;                                                                            *
;   To place this PSECT at this address the following option                 *
;   is inserted on the Additional Command Line Options for the               *
;   target filename hex node.                                                *
;                                                                            *
;   -L-Paccesscode=1FFAh                                                     *
;                                                                            *
;*****************************************************************************

    psect accesscode,class=CODE,delta=2
    global _master_code


_master_code:

    retlw 0x01                          ; 1st digit of code
    retlw 0x02                          ; 2nd digit of code
    retlw 0x03                          ; 3rd digit of code
    retlw 0x04                          ; 4th digit of code
    retlw 0x00                          ; null character for string

    end
```

```
/*********************************************************************
*                                                                    *
*   Filename:       hcs515ee.h                                       *
*   Date:           07/18/99                                         *
*   File Version:   1.00                                             *
*                                                                    *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                 *
*                                                                    *
*********************************************************************/


#define USER_CODE    0x80
#define MSTR_CODE    0x86                  // define EE address for Master Code entry
#define XMTR_CNT     0xBC                  // define EE address learned XMTR count

#define BT_ON_CNT    0xF0                  // define EE address for battery cycle count
#define BT_ON_HRS    0xF1                  // define EE address for battery on TOD
#define BT_OFF_HRS   0xF4                  // define EE address for battery off TOD

#define ALRM_HRS     0xF8                  // define EE address for Alarm TOD
#define ALRM_STAT    0xFB                  // define EE address for Alarm Status
#define LAST_XMIT    0xFF                  // define EE address for last XMTR type


#define LOW          0x55                  // define byte for label "LOW"
#define HIGH         0xAA                  // define byte for label "HIGH"

#define ALRM         0x41                  // define byte for label "ALRM"
#define CLEAR        0xBE                  // define byte for label "CLEAR"
#define TST_LRN      0xB1                  // define byte for label "TST_LRN"

#define OKAY         0x4F                  // define byte for label "OKAY"
#define ERROR        0xB0                  // define byte for label "ERROR"
```

Please check Microchip's Worldwide Website at www.microchip.com for the latest revisiion of the source code.

APPENDIX C: SENSOR MODULE CODE FILES

```
/************************************************************************
*                                                                       *
*   Wireless Home Security with Keeloq and the PICmicro                 *
*                                                                       *
************************************************************************
*                                                                       *
*   Filename:       sensor08.c                                          *
*   Date:           05/24/99                                            *
*   File Version:   1.00                                                *
*                                                                       *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3                    *
*                                                                       *
*   Author:         Richard L. Fischer                                  *
*   Company:        Microchip Technology Incorporated                   *
*                                                                       *
************************************************************************
*                                                                       *
*   Files required:                                                     *
*                                                                       *
*                   pic.h         (Hi-Tech file)                        *
*                   cnfig50x.h                                          *
*                                                                       *
*                   sensor08.c    (Sensor Module code)                  *
*                   powerup08.as  (Hi-Tech file, modified)              *
*                   delay12.as    (Delay Routine code)                  *
*                                                                       *
************************************************************************
*                                                                       *
*   Notes:                                                              *
*                                                                       *
*   PICmicro -> PIC12C508A                                              *
*   Fosc -> Internal RC                                                 *
*   Weak-pull-ups disabled (current constraints)                        *
*   Wakeup from sleep --> typical 400uS                                 *
*   Wake-up from sleep until test of GPWUF --> typical 20.03mS          *
*   Current consumption in sleep mode:  (sleep duration -> ~100sec)     *
*                                  -> ~3.2uA @ 6.4Vdc (new battery)     *
*                                  -> ~2.0uA @ 3.3Vdc (battery EOL)     *
*                                                                       *
*   Sensor operational lower-limit voltage:                             *
*                              -> 4Vdc (Low voltage status to base)     *
*                                                                       *
*                                                                       *
*   Current consumption during wakeup:                                  *
*                       housekeeping only: (typical 56mS)               *
*                                  -> ~ 0.70mA @ 6.4Vdc                 *
*                                  -> ~ 0.30mA @ 3.3Vdc                 *
*                       intentional radiation: (typical 700mS)          *
*                                  -> ~ 4.64mA @ 6.4Vdc                 *
*                                  -> ~ 2.22mA @ 3.3Vdc                 *
*                                                                       *
*                                                                       *
* Memory Usage Map:                                                     *
*                                                                       *
* User segment  $0000 - $0002  $0003 (   3) bytes total User segment    *
* Program ROM   $0000 - $01FE  $01FF ( 511) words                       *
* Program ROM   $0FFF - $0FFF  $0001 (   1) words                       *
*                              $0200 ( 512) words total Program ROM     *
*                                                                       *
* Bank 0 RAM    $0007 - $000C  $0006 (   6) bytes total Bank 0 RAM      *
*                                                                       *
************************************************************************/

#include <pic.h>                            // processor if/def processor file
#include "cnfig50x.h"                       // configuration bit definitions


  __CONFIG   (CONBLANK & MCLRE_OFF & CP_OFF & WDT_OFF & IntRC_OSC);


extern void Delay_Ms_4MHz(char delay);      //prototype for delay function

// MACROS DEFINED HERE
#define  PUT_COUNT   54                     // define number of PUT fires on pin GP0
                                            // before initiating sensor okay signal
                                            // each PUT fire sequence is 100 seconds,
                                            // total time is therefore 1.5hrs
```



```
#define  SLEEP   asm ("sleep")              // macro for sleep instruction
                                            // RF and begin PUT discharge
#define  NOP     asm ("NOP");               // power-up stabilization period (3uS)



#define PortBit(port,bit)    ((unsigned)&(port)*8+(bit))
static bit AUTO_CYCLE   @ PortBit(GPIO,0);   // define auto cycle pin
static bit LEARN_TEST   @ PortBit(GPIO,1);   // define okay/learn/test pin
static bit ALARM_SIGNAL @ PortBit(GPIO,3);   // define alarm signal pin


// GLOBALS DEFINED HERE
persistent char auto_transmit;               // variable used for counting PUT based
                                             // wake-ups


//*********************************************************************
//*********************************************************************
/* Determine source of wake-up from SLEEP. Wake-up source is either:

   1. Alarm sensor trip via active high on pin GP3.
   2. Auto wake-up via Programmable Unijunction Transistor (PUT)
      firing active high on pin GP0.
   3. Learn/test active high on pin GP1.

*/

void Identify_wakeup(void)
{
   if (ALARM_SIGNAL)                         // has alarm sensor been activated?
   {
     GPIO = 0b111010;                        // set GP5 pin state to turn on HCS200 & RF
                                             // also set GP2 for PUT discharge state
     TRIS = 0b011011;                        // set GP5 as output, power up HCS200 and
                                             // RF and begin PUT discharge
     NOP;                                    // power-up stabilization period (3uS)
     NOP;
     NOP;

     TRIS = 0b001011;                        // set GP4 as output, assert S0 on HCS200
     Delay_Ms_4MHz(255);                     // ensure multiple alarm transmissions
     Delay_Ms_4MHz(255);                     // ensure multiple alarm transmissions
     Delay_Ms_4MHz(80);                      // ensure multiple alarm transmissions
                                             // want 5 transmissions at 3.3Vdc @ 4MHz

     auto_transmit = 0;                      // reset auto cycle counter since alarm
   }                                         // tripped

   else if (LEARN_TEST)                      // learn/test condition entered?
   {
     GPIO = 0b111010;                        // set GP5 pin state to turn on HCS200 & RF
                                             // also set GP2 for PUT discharge state

     TRIS = 0b011011;                        // set GP5 as output, power up HCS200 and
                                             // RF and begin PUT discharge
     NOP;                                    // power-up stabilization period (3uS)
     NOP;
     NOP;

     TRIS = 0b011001;                        // set GP1 as output, assert S1 on HCS200
     Delay_Ms_4MHz(120);                     // ensure single transmission time
                                             // want 1 transmission at 3.3Vdc @ 4MHz
     auto_transmit = 0;                      // reset auto cycle counter since learn/test
                                             // has been activated
   }

   else                                      // else PUT fired (Vgs = 0Vdc)
   {
     if (++auto_transmit < PUT_COUNT)        // time for initiating sensor okay?
     {
       GPIO = 0b011010;                      // set GP2 and GP5 pin latches to logic low
       TRIS = 0b011011;                      // set GP2 and GP5 pin direction to outputs
                                             // begin PUT discharge immediately
       Delay_Ms_4MHz(30);                    // PUT discharge time period (minimum)
     }

     else                                    // else it is time for initiating sensor okay?
     {
       GPIO = 0b111010;                      // set GP5 pin state to turn on HCS200 & RF
                                             // also set GP2 for PUT discharge state
       TRIS = 0b011011;                      // set GP5 as output, power up HCS200 and RF
```

```
                                        // and begin PUT discharge
       NOP;                             // power-up stabilization period (3uS)
       NOP;
       NOP;

       TRIS = 0b001001;                 // set GP1/GP5 as output, assert S0/S1 on HCS200
       Delay_Ms_4MHz(255);              // ensure multiple sensor okay transmissions
       Delay_Ms_4MHz(10);               // ensure multiple sensor okay transmissions
                                        // want 2 transmissions at 3.3Vdc @ 4MHz (worst case)
       auto_transmit = 0;               // reset auto cycle counter
     }
   }
}


void main(void)
{

   if (GPWUF)                           // has wake-up on pin-change occurred?
   {
     OPTION = 0b11011111;               // enable GP2 for I/O

     GPIO = 0b011110;                   // GPIO pin latch state is not affected by
     TRIS = 0b011011;                   // wake-up, keep I/O direction same as in
                                        // sleep to ensure valid read of Vih on pin

     Delay_Ms_4MHz(20);                 // short delay for debounce and pin state
                                        // stabilization

     Identify_wakeup();                 // determine source of wake-up from sleep

     GPIO = 0b011010;                   // set GP2 & GP5 pin latches to logic low
     TRIS = 0b011011;                   // power off to HCS200 and RF
     GPWUF = 0;                         // reset wake-up status bit
   }

   else if (TO && PD)                   // else, is it a power up condition?
   {
     auto_transmit = 0;                 // initialize wake-up counter, powerup only
   }

                                        // code to do all the time independant of reset
                                        // condition (GPWUF or Power-up)

   OPTION = 0b01011111;                 // enable wake-up on change and GP2 I/O
   GPIO = 0b011010;                     // set GP2 to logic low, PUT discharge state
   TRIS = 0b011011;                     // set GP0 and GP1 for inputs, GP2 output
                                        // power off to HCS200 and RF

   GPIO = 0b011110;                     // start PUT cycle via release of pin GP2
   GPIO;                                // dummy read prior to entering sleep
   SLEEP;                               // go to sleep

 #asm
   rept 0x165
   goto 0x1FF                           // fill unused memory with goto 0x1FF
   endm
 #endasm

}
```

```
;*********************************************************************
;                                                                    *
;   Wireless Home Security with Keeloq and the PICmicro              *
;                                                                    *
;*********************************************************************
;                                                                    *
;   Filename:       delay12.as                                       *
;   Date:           07/18/99                                         *
;   File Version:   1.00                                             *
;                                                                    *
;*********************************************************************
;                                                                    *
;   Notes:                                                           *
;                                                                    *
;   The file is contains the assembly code for executing intervals   *
;   of a 1mS delay. The timing is based upon the PIC12C508A          *
;   internal RC oscillator of 4MHz.                                  *
;                                                                    *
;*********************************************************************

        psect   text0,class=CODE,local,delta=2
        global_Delay_Ms_4MHz                ; make function scope global
        signat_Delay_Ms_4MHz,4216           ; signature for link time

extrn   string_table

_Delay_Ms_4MHz
        nop
        movwf   ((?a_Delay_Ms_4MHz+0) &0x7F)  ; init outerloop count
outer:
        movlw   0xF9
        movwf   ((?a_Delay_Ms_4MHz+1) &0x7F)  ; init innerloop count
inner:
         nop
         decfsz  ((?a_Delay_Ms_4MHz+1) &0x7F) ; decrement innerloop count
         goto    inner
         decfsz  ((?a_Delay_Ms_4MHz+0) &0x7F) ; decrement outerloop count
         goto    outer

        movf    0x08,w                      ; restore w for jump table access
        goto    string_table                ; go to string table

FNSIZE _Delay_Ms_4MHz,2,1                   ; inform linker of argument and local variable
                                            ; sizes for function
global  ?a_Delay_Ms_4MHz                    ; declare symbol as public

        end                                  end assembly
```

```
;*****************************************************************************
;                                                                            *
;   Wireless Home Security with Keeloq and the PICmicro                      *
;                                                                            *
;*****************************************************************************
;                                                                            *
;   Filename:       powrup08.as                                              *
;   Date:           07/18/99                                                 *
;   File Version:   1.00                                                     *
;                                                                            *
;*****************************************************************************
;                                                                            *
;                                                                            *
;                                                                            *
;*****************************************************************************


    #include"sfr.h"

    global  powerup,start
;   psect   powerup,class=CODE,delta=2

extrn  _main

powerup
#if defined(_12C508) || defined(_12C509)
    movwf   5       ;store calibration to OSCCAL
    goto    _main
#endif
#if defined(_PIC14)
    clrf    STATUS
    movlw   start>>8
    movwf   PCLATH
    goto    start & 7FFh
#endif
#if defined(_PIC16)
    movlw   start>>8
    movwf   PCLATH
    movlw   start & 0xFF
    movwf   PCL
#endif
    end     powerup
```

```
/*****************************************************************
*                                                                *
*   Filename:       cnfig50x.h                                   *
*   Date:           07/18/99                                     *
*   File Version:   1.00                                         *
*                                                                *
*   Compiler:       Hi-Tech PIC C Compiler V7.83 PL3              *
*                                                                *
*****************************************************************/

/*****  CONFIGURATION BIT DEFINITIONS FOR PIC12C508(A) and PIC12C509(A) PICmicro  *****/


#define    CONFIGADD    0xFFF
#define    CONBLANK     0xFFF

#define    MCLRE_ON     0xFFF
#define    MCLRE_OFF    0xFEF
#define    CP_ON        0xFF7
#define    CP_OFF       0xFFF
#define    WDT_ON       0xFFF
#define    WDT_OFF      0xFFB
#define    LP_OSC       0xFFC
#define    XT_OSC       0xFFD
#define    IntRC_OSC    0xFFE
#define    ExtRC_OSC    0xFFF
```

TB021

A Guide to Designing for EuroHomelink® Compatibility

Author: *Kobus Marneweck*
Microchip Technology Inc.

INTRODUCTION

The Prince EuroHomelink system is an in-car RF control system to control gate and garage door openers, door locks, and home security systems. Although the system can support older generation fixed code systems all new systems require code hopping. The KEELOQ® code hopping system from Microchip Technology is the default code hopping system in the EuroHomelink. A standardized system allows different manufacturers to make EuroHomelink compatible receivers while still allowing each manufacturer to implement unique features and maintain control over their own manufacturer's code. This document sets out the requirements for a EuroHomelink compatible system.

REFERENCED DOCUMENTS

- *An Introduction to KEELOQ Code Hopping* – DS91002
- *HCS200 KEELOQ Code Hopping Encoder* Data Sheet – DS40138
- *HCS300 KEELOQ Code Hopping Encoder* Data Sheet – DS21137
- *HCS301 KEELOQ Code Hopping Encoder* Data Sheet – DS21143
- *HCS360 KEELOQ Code Hopping Encoder* Data Sheet – DS40152
- *HCS361 KEELOQ Code Hopping Encoder* Data Sheet – DS40146
- *HCS410 KEELOQ Code Hopping Encoder* Data Sheet – DS40158
- *Secure Learning RKE systems using KEELOQ Encoders* – DS91000
- *Guidelines for KEELOQ Secure Learning Implementation* – DS91007_C
- *PICmicro™ Midrange MCU Code Hopping Decoder* – DS00672

SYSTEM DESCRIPTION

A EuroHomelink system consists of a receiver that can accept KEELOQ code hopping transmissions from an in-car EuroHomelink transmitter or the manufacturer's own transmitters. See the referenced documents for details on the KEELOQ code hopping system.

Learning

The learning system used is the KEELOQ Secure Learning system. Technical Briefs DS9100 and DS91007_C describe how Secure Learning works in more detail. In a Secure Learning system a seed value is transmitted during the training or learning phase. This seed is used to calculate a encryption/decryption key for the transmitter using a key generation algorithm and a manufacturer's code. The system makes provision for a common easy-train default manufacturer's code and a private manufacturer's code used to generate keys for the manufacturer's own transmitters.

Easy-train default Manufacturer's Code

Microchip Technology will act as an independent custodian of the easy-train default manufacturer's code. Each manufacturer can also choose their own private manufacturer's code. In order for Microchip to act as an independent custodian, all decoders or microcontrollers used in receivers must be loaded by Microchip with the easy-train default manufacturer's code.

Serial Number Allocation

The EuroHomelink system will use a 28-bit encoder serial number. The serial number space will be allocated to allow the EuroHomelink transceiver to detect the specific manufacturer.

TABLE 1: SERIAL NUMBER ALLOCATION

Start	End	Size	Manufacturer
0000000H	0000000H	1	Test transmitter
0000001H	0FFFFFFH	16M	Manufacturer 1
1000000H	1FFFFFFH	16M	Manufacturer 2
2000000H	2FFFFFFH	16M	Manufacturer 3
3000000H	3FFFFFFH	16M	Manufacturer 4
4000000H	4FFFFFFH	16M	Manufacturer 5
5000000H	5FFFFFFH	16M	Manufacturer 6
6000000H	6FFFFFFH	16M	Manufacturer 7
7000000H	7FFFFFFH	16M	Manufacturer 8
8000000H	8FFFFFFH	16M	Manufacturer 9
9000000H	9FFFFFFH	16M	Manufacturer 10
A000000H	AFFFFFFH	16M	Manufacturer 11
B000000H	FFFFFF0H	64M	Easy-train default
FFFFFFEH	FFFFFFEH	1	EuroHomelink Test TX 1 – Normal
FFFFFFFH	FFFFFFFH	1	EuroHomelink Test TX 2 - Secure

Note: Microchip offers Serialized Quick Turn Programming™ (SQTP) service to preprogram encoder devices. Software is provided to the customer to generate code files in a secure format. Files start on 50,000 boundaries and are 50,000 in size.

SQTP is a service mark of Microchip Technology Inc.

RECEIVER REQUIREMENTS

Modulation Format

The standard KEELOQ PWM (1/3, 2/3) format will be used. The code word consists of a preamble, header, 32 bit encrypted portion, 28 bit serial number, 4 bit function code, a battery low indication bit and a repeat or CRC bit depending on the encoder used. The last bit will be ignored by the receiver.

Bit Rate and Bandwidth

An elemental time period of TE = 200 µs is used. This translates to a bit rate of 1.67 KBps. The receiver bandwidth need to be at least 3.3 KHz.

Learning Procedure

The learning procedure consists of placing the receiver in a learn mode by any of the procedures described in DS91007_C. Push the button that will be used for that gate or door. The EuroHomelink transceiver will transmit a code hopping transmission for 3 seconds followed by the seed transmission for 3 seconds from which the key is calculated. The code hopping and seed transmission will alternate every 3 seconds. A successful learn can be confirmed by pressing the appropriate button to open or close the gate or door.

Reference Decoder Application Note

The *PICmicro Midrange MCU Code Hopping Decoder* application note implements a EuroHomelink compatible receiver on a PIC16C61. This application note and code can be used as the basis to implement an integrated receiver and motor control system.

Other Decoder Solutions

Microchip will make available a dedicated decoder that will implement a dual key and will be EuroHomelink compatible. This decoder can be used in conjunction with a microcontroller to implement a system.

TRANSMITTER REQUIREMENTS

TABLE 2: SUMMARY OF ENCODERS AND THEIR FEATURES

Encoder	HCS200	HCS300	HCS301	HCS360	HCS361	HCS410
Inputs	3	4	4	4	4	3
Functions	7	15	15	15	15	7
Voltage	3.5-13V	2.0-6.3V	3.5-13V	2.0-6.6V	2.0-6.6V	2.0-6.6V
Serial # bits	28	28	28	28/32	28/32	28/32
Key bits	64	64	64	64	64	64
Transmission	66	66	66	67	67	69
Queuing bits	0	0	0	0	0	2
Seed	32	32	32	48	48	60
Baud rates	2	3	3	2	2	3
LED output	No	Yes	Yes	Yes	Yes	Yes
Time-out	Yes	Select	Select	Select	Select	Select
Modulation	PWM	PWM	PWM	PWM Manchester	PWM VPWM	PWM Manchester
Application	Low cost/ Low end	Mid-range	Mid-range	High-end OEM	High-end OEM	High-end Transponder

TABLE 3: ENCODER CONFIGURATION OPTIONS

Option	Setting	Relevant Encoder
Discrimination bits	LSB 10 or 12 bits of Serial number	All
Overflow	0 or 00	All
Low voltage trip point	0 or 1 depending on battery voltage	All
Obdurate	T_E = 200 µs	All
Envelope Encryption	Off	HCS300, HCS301
Seed transmission	Enabled	HCS360, HCS361, HCS410
Limited seed	Not specified	HCS360, HCS361, HCS410
Extended serial number	Disabled	HCS360, HCS361, HCS410
Independent mode	Disabled	HCS360, HCS361
Long Guard time	Disabled	HCS360
PWM select	Enabled	HCS360, HCS361, HCS410
Delay mode	Disabled	HCS360, HCS361
USR bits	00	HCS360, HCS361
Time out	Enabled	All
Blank Alternate Code Word	Disabled	HCS360, HCS361, HCS410
Minimum 3 Code words	Not specified	HCS410
Delayed counter increment	Disabled	HCS410

TABLE 4: SEED TRANSMISSION OPTIONS ON ENCODERS

S[3210]	Encoder	Notes
X111	HCS200, HCS410	
1111	HCS300, HCS301	
0011	HCS360, HCS361, HCS410	Delayed after 3 seconds
1001	HCS360, HCS361	

EUROHOMELINK OPERATION

Training Procedure

The EuroHomelink is factory set to default to KEELOQ using the easy-train default manufacturer's code. No training is therefore, required by the EuroHomelink system and the receiver's learning procedure can be followed.

For a non-default manufacturer's key or fixed code system, the EuroHomelink must be trained. Press the outer buttons on the EuroHomelink transceiver for 20 seconds. Release the buttons when the LED starts blinking rapidly. Press any button (the LED will blink slowly) and operate the transmitter. Successful training will be indicated by a rapid blinking LED. The EuroHomelink is now ready to be trained to the transceiver.

Function Code During Seed Transmission

The function code is set to 1111. On KEELOQ encoders this may be different.

Bit Rate

An elemental time period of T_E = 200 µs is used. This translates to a bit rate of 1.67 KBps.

Frequency

The operating frequency will be:

433.92 MHz ± 100 KHz

AM Modulation Depth

3 dB (not on/off modulated)

SYSTEM VALIDATION

Test Transmitter and Test Modes

A serial number 0000000H is reserved as a test transmitter. Manufacturers can program a transmitter with serial 0000000H and their manufacturer's code to test their receiver.

The EuroHomelink accommodates two test transmitters. The table shows the configuration of the test transmitters. This data will be used to define the output transmissions of the EuroHomelink after being trained to one of the two serial numbers identified in Table 1. A default manufacturer's code of 01234567-89ABCDEFh is used.

TABLE 5: TEST TRANSMITTER AND TEST MODES

Test transmitter	Serial Number	Seed	Key
1. Normal Learn (Decrypt)	1234567h	Not applicable	0516FBE9-89074278h
2. Secure Learn (XOR)	1234567h	0123456789	00000000-00000000h
Learn (XOR)		ABCDEFh	

Reference Test Receiver

The application note *PICmicro Midrange MCU Code Hopping Decoder* can be used a reference receiver. Pre-programmed PIC16C61s microcontrollers can be ordered from Microchip technology with the easy-train default manufacturer's code embedded.

TECHNICAL CONTACTS

Sharon Mathias – Prince UK

Tel: UK 1-926-333-035
email: sharon.mathias@jci.com

Lothar Grad – Price Germany

Tel: GER 7031-9391-46
email: lothar.grad@jci.com

NOTES:

SECTION 4
ANALOG/INTERFACE PRODUCT APPLICATION NOTES AND TECHNICAL BRIEFS

AN679

Temperature Sensing Technologies

Author: *Bonnie Baker*
Microchip Technology Inc.

INTRODUCTION

Of all of the sensing technologies, temperature sensing is the most common. This phenomena can be explained by citing examples in a multitude of applications where knowing and using the actual or relative temperature is critical. For instance, other sensors such as pressure, force, flow, level, and position many times require temperature monitoring in order to insure accuracy. As an example, pressure and force are usually sensed with resistive Wheatstone bridge configurations. The temperature errors of the resistive elements of these bridges can exceed the actual measurement range of the sensor, making the pressure sensor's output fairly useless, unless the temperature of the bridge is known. Flow and level sensor accuracies are dependent on the density of the liquid or gas.

One variable that affects the accuracy of these sensors is the temperature of that material. Position is most typically used in motor control. In these circuits, temperature affects the efficiency of the motor. Consequently, the understanding of temperature sensing is needed in order to fully understand how to accurately sense most other physical phenomena.

This application note will cover the most popular temperature sensor technologies to a level of detail that will give the reader insight into how to determine which sensor is most appropriate for the application. This note is written from the perspective of catering to the complex issues of the sensing environment and required accuracy. Once the sensor is selected, subsequent Microchip application notes can be used to design appropriate microcontroller interface circuits. These circuits will offer the complete signal path from the low level output signals of the sensor, through the analog signal conditioning stages to the microcontroller. Techniques such as sensor excitation, sensor signal gain, and digital linearization are reserved for these further discussions.

SO MANY TEMPERATURE SENSORS

The most popular temperature sensors used today are the Thermocouple, Resistive Temperature Device (RTD), Thermistor, and the newest technology, the Integrated Silicon Based Sensors. There are other sensing technologies, such as Infrared (Pyrometers) and Thermal Pile. These alternatives are beyond the scope of this application note.

Each of these sensor technologies cater to specific temperature ranges and environmental conditions. The sensor's temperature range, ruggedness, and sensitivity are just a few characteristics that are used to determine whether or not the device will satisfy the requirements of the application. No one temperature sensor is right for all applications. The thermocouple's wide temperature range is unrivalled as is the excellent linearity of the RTD and the accuracy of the Thermistor.

Table 1 summarizes the main characteristics of these four temperature sensors. This table can be used during the first pass of the sensor selection process. Further details concerning the construction and characteristics of these sensors are given in the following sections of this application note.

To complement the specifications sited in Table 1, a list of typical applications for these four temperature sensors are shown in Table 2.

	Thermocouple	RTD	Thermistor	Integrated Silicon
Temperature Range	–270 to 1800°C	–250 to 900 °C	–100 to 450°C	-55 to 150°C
Sensitivity	10s of μV / °C	0.00385 Ω / Ω / °C (Platinum)	several Ω / Ω / °C	Based on technology that is -2mV/°C sensitive
Accuracy	±0.5°C	±0.01°C	±0.1°C	±1°C
Linearity	Requires at least a 4th order polynomial or equivalent look up table.	Requires at least a 2nd order polynomial or equivalent look up table.	Requires at least 3rd order polynomial or equivalent look up table.	At best within ±1°C. No linearization required.
Ruggedness	The larger gage wires of the thermocouple make this sensor more rugged. Additionally, the insulation materials that are used enhance the thermocouple's sturdiness.	RTDs are susceptible to damage as a result of vibration. This is due to the fact that they typically have 26 to 30 AWG leads which are prone to breakage.	The thermistor element is housed in a variety of ways, however, the most stable, hermetic Thermistors are enclosed in glass. Generally thermistors are more difficult to handle, but not affected by shock or vibration.	As rugged as any IC housed in a plastic package such as dual-in-line or surface outline ICs.
Responsiveness in stirred oil	less than 1 Sec	1 to 10 Secs	1 to 5 Secs	4 to 60 Secs
Excitation	None Required	Current Source	Voltage Source	Typically Supply Voltage
Form of Output	Voltage	Resistance	Resistance	Voltage, Current, or Digital
Typical Size	Bead diameter = 5 x wire diameter	0.25 x 0.25 in.	0.1 x 0.1 in.	From TO-18 Transistors to Plastic DIP
Price	$1 to $50	$25 to $1000	$2 to $10	$1 to $10

TABLE 1: *The most common temperature sensors in industry are the thermocouple, RTD, thermistor, and integrated silicon based. No one temperature sensor is right for all applications. The thermocouple's wide temperature range is unrivalled as is the excellent linearity of the RTD and the accuracy of the thermistor. The silicon sensor is easy to implement and install in a circuit.*

Sensor Type	Application
Thermocouple	Extremely high temperature sensing, biophysics, metal cutting research, gas chromatography, internal combustion engine temperatures, chemical reactions
RTD	Cold junction compensation, bridge temperature, calibration, process control.
Thermistor	Cold junction compensation, bridge temperature sensing, pyrometer calibration, vacuum manometers, anemometers, flow meters, liquid level, fluid velocity, thermal conductivity cells, gas chromatography
Silicon Based	Cold junction compensation, personal computers, office electronics, cellular phones, HVAC, battery management, four speed controls

TABLE 2: *Listed are some examples of the applications that each temperature sensor is best suited for.*

THE VERSATILE, INEXPENSIVE THERMOCOUPLE

The thermocouple consists of two wires of dissimilar metals that are soldered together at one end as shown in Figure 1. The temperature at the Reference Junction (also know as the Cold Junction Compensation Point) is used to negate the errors contributed by the Iron-Copper and Constantan-Copper junctions. The connecting point of the two metals of the thermocouple is positioned on the target where the temperature measurement is needed.

This configuration of materials produces a voltage between the two wires at the unsoldered end that is a function of the temperature of all of the junctions. Consequently, the thermocouple does not require voltage or current excitation. As a matter of fact, an attempt to provide either type of excitation could introduce errors into the system.

Since a voltage develops at the open end of the two dissimilar wires, it would seem as if the thermocouple interface could be done in a straight forward manner by measuring the voltage difference between the wires.

This could easily be the case if it wasn't for the fact that the termination ends of the thermocouple wires connect to another metal, usually copper.

This creates another pair of thermocouples, which introduces a significant error to the system. The only way to negate this error is to sense the temperature at the Reference Junction box (Figure 1) and subtract the contributing errors of these connections in a hardware solution or a combination of software and hardware.

Pure hardware calibration techniques are more limited in terms of linearization correction than the combination of software and hardware techniques. Typically, an RTD, Thermistor, or Integrated Silicon Sensor is used to sense this junction temperature accurately.

In principle the thermocouple can be made from any two metals, however, in practice standard combinations of these two metals have been embraced because of their desirable qualities of linearity and their voltage magnitude drop versus temperature. These common thermocouple types are E, J, T, K, N, S, B, and R (summarized in Table 3 and Figure 2).

Thermocouples are highly non-linear and require significant linearization algorithms, as will be discussed later. The Seebeck Coefficient in Table 3 represents the average drift of the specific thermocouple at a specific temperature.

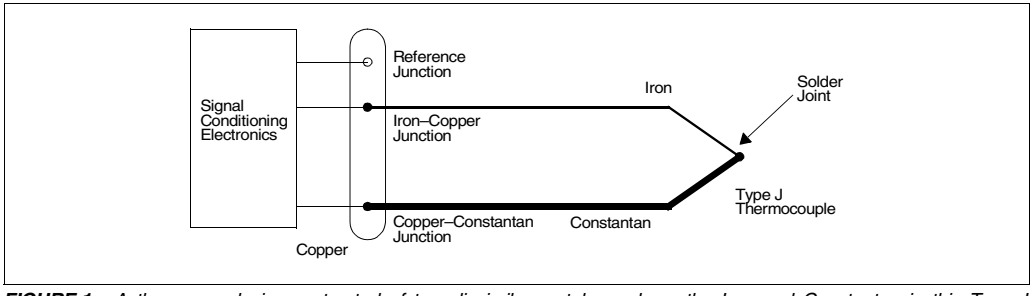


FIGURE 1: *A thermocouple is constructed of two dissimilar metals, such as the Iron and Constantan in this Type J thermocouple. The temperature of the Reference Junction Compensation (also known as the Cold Junction Compensation or Isothermal Block) is used to negate the errors contributed by the Iron-Copper and Constantan-Copper Junctions.*

.

Thermocouple Type	Conductors	Temperature Range (°C)	Seebeck Coefficient	Application Environments
E	Chromel, Constantan	-200 to 900	60µV/°C	oxidizing, inert, vacuum
J	Iron, Constantan	0 to 760	51µV/°C	vacuum, oxidizing reducing, inert
T	Copper, Constantan	–200 to 371	40µV/°C	corrosive, moist, subzero
K	Chromel, Alumel	-200 to 1260	40µV/°C	completely inert
N	Nicrosil, Nisil	0 to 1260	38µV/°C	oxidizing
S	Platinum(10% Rhodium), Platinum	0 to 1480	11µV/°C	oxidizing, inert
B	Platinum (30% Rhodium) Platinum (6% Rhodium)	0 to 1820	8µV/°C	oxidizing, inert
R	Platinum (13% Rhodium), Platinum	0 to 1480	12µV/°C	oxidizing, inert

TABLE 3: *The most common thermocouple types are shown with their standardized material and performance specifications. These thermocouple types are fully characterized by the American Society for Testing and Materials (ASTM) and specified in IST-90 units per NIST Monograph 175.*

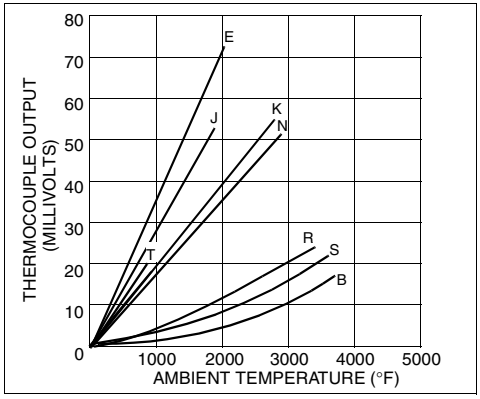


FIGURE 2: *Thermocouples are sensitive to a wide range of temperatures making them appropriate for a variety of hostile environments.*

At the time of shipment, the thermocouple performance is guaranteed by the vendor in accordance with NIST 175 standards (adopted by ASTM). These standards define the temperature behavior of the thermocouple as well as the quality of the material used.

Thermocouples are extremely non-linear when compared to RTD, Thermistor, and Integrated Silicon Sensors. Consequently, complex algorithms must be performed with the processor portion of the circuit. An example of the complexity of the calculation is shown in Table 4. These are the Type K Thermocouple coefficients that can be used to linearize the output voltage results for a temperature range of 0°C to 1372°C. These coefficients are used in the equation

$$V = c_0 + c_1 t + c_2 t^2 + c_3 t^3 ...$$

where

V is equal to the voltage across the thermocouple junction, and

t is equal to the temperature.

c_0	$-1.7600413686 \times 10^{-2}$
c_1	$3.8921204975 \times 10^{-2}$
c_2	$1.8558770032 \times 10^{-5}$
c_3	$-9.9457592874 \times 10^{-8}$
c_4	$3.1840945719 \times 10^{-10}$
c_5	$-5.6072844889 \times 10^{-13}$
c_6	$5.6075059059 \times 10^{-16}$
c_7	$-3.2020720003 \times 10^{-19}$
c_8	$9.7151147152 \times 10^{-23}$
c_9	$-1.2104721275 \times 10^{-26}$

TABLE 4: *These are the Type K thermocouple coefficients that can be used to linearize the output voltage results for a temperature range of 0°C to 1372°C. These coefficients are used in the equation* $V = c_0 + c_1 t + c_2 t^2 + c_3 t^3 ...$ *where V is equal to the voltage across the thermocouple junction, and t is equal to the temperature.*

The alternative to using these complex calculations is to use program memory for a look-up table. The replacement look-up table for the equation coefficients of the Type K thermocouple in Table 4 is approximately an 11 x 14 array of decimal integers ranging from 0.000 to 13.820.

Additionally, the thermocouple can quantify temperature as it relates to a reference temperature. The reference temperature is defined as the temperature at the end of the thermocouple wires furthest from the soldered bead. This reference temperature is usually sensed using an RTD, Thermistor, or Integrated Silicon Sensor.

The thermal mass of the thermocouple is smaller than the RTD or Thermistor, consequently the response of the thermocouple as compared to larger temperature sensors is faster. The wide temperature ranges of the sensor makes it exclusively appropriate for many hostile sensing environments.

Thermocouple Error Analysis

Thermocouples are generally low cost, rugged and available in smaller sizes than the other temperature sensors. Any stress on the material due to bending stretching or compression can change the characteristics of the thermal gradients. Additionally, corrosive material can penetrate the insulation material and cause a change in the thermal characteristics. It is possible to encase the thermocouple bead in protective tubing such as a ceramic tube for high temperature protection. Metallic wells can also provide mechanical protection.

The thermocouple voltage drop occurs along the temperature gradient down the length of the two dissimilar metals. This does not imply that shorter versus longer wires will necessarily have differing Seebeck Coefficients. With shorter wires, the temperature gradient is simply steeper. However, the longer wires do have an advantage in terms of conduction affects. With the longer wires the temperature gradient is lower and conduction losses are reduced.

On the down side, these types of temperature sensors have a very low output signal. This places additional requirements on the signal conditioning circuitry that follows the thermocouple. In addition to this low level output signal, the linearity of the device requires a considerable amount of calibration. This calibration is typically done in firmware as well as software. In firmware, an absolute temperature reference is needed which serves as a “cold junction” reference. In software, the linearity errors of the thermocouple are reduced with look-up tables or high order polynomial equations. And finally, EMI signals are easily coupled in to this two-wire system.

Lower gage wires are required for higher temperatures and will also have a longer life. However, if sensitivity is a prime concern, larger wire gages will provide better measurement results.

To summarize, thermocouples are usually selected because for the wide temperature range, ruggedness, and price. Accuracy and good linearity are hard to achieve in precision systems. If high accuracy is desirable, other temperature sensors may be a better alternative.

THE RTD IS ABSOLUTELY AN ALTERNATIVE

RTD element technologies are constantly improving, enhancing the quality of the temperature measurement. To produce a high quality, accurate temperature measurement system, the selection of the RTD element is critical. The RTD (Resistance Temperature Detector) is a resistive element constructed from metals, such as, Platinum, Nickel or Copper. The particular metals that are chosen exhibit a predictable change in resistance with temperature. Additionally, they have the basic physical properties that allow for easy fabrication. The temperature coefficient of resistance of these metals is large enough to render measurable changes with temperature.

Other temperature sensing devices, such as thermocouples, fall short of giving the designer an absolute result that is fairly linear over temperature. The linear relation between resistance and temperature of the RTD simplifies the implementation of signal conditioning circuitry. The resistance change to temperature of each of these types of RTDs is shown in Table 5. Platinum RTDs (PRTD) are the most accurate and reliable of the three types shown in Table 5.

Of all the material types, Platinum RTDs are best suited for precision applications where absolute accuracy and repeatability is critical. The platinum material is less susceptible to environmental contamination, where copper is prone to corrosion causing long term stability problems. Nickel RTDs tolerate environmental conditions fairly well, however, they are limited to smaller temperature ranges.

The PRTD has nearly linear thermal response, good chemical inertness and is easy to manufacture in the form of small-diameter wires or films. As shown in Table 5, the resistivity of the platinum is higher than the other metals, making the physical size of the element smaller. This offers advantages where "real-estate" is at a premium as well better thermal responsiveness.

Thermal responsiveness of an RTD affects the measurement time. It is also dependent on the housing material of the RTD and the size of the implementation of the RTD element. Elements with smaller dimensions can be housed in smaller packages. Since RTD are typically smaller, their thermal response times can be shorter than silicon based temperature sensors.

The absolute, 0°C value of the element is available in a wide range of resistances and can be specified by the user. For instance, the standard resistance of a platinum RTD (PRTD) is 100Ω. But, they are also available as 50, 100, 200, 500 1000 or 2000Ω elements.

As stated before, the RTD is an absolute temperature sensing devices as opposed to the thermocouple, which senses relative temperatures. Consequently, additional temperature sensors would not necessarily enhance the accuracy of the system.

RTD Detector Material	Thermal Response (at 0°C)	Typical Material Resistivity (at 0°C)
Platinum	0.00385 Ω/Ω/°C (IEC 751)	9.81 x 10^{-6} Ω cm
Nickel	0.00672 Ω/Ω/°C	5.91 x 10^{-6} Ω cm
Copper	0.00427 Ω/Ω/°C	1.53 x 10^{-6} Ω cm

TABLE 5: *RTD temperature sensing devices are available in a variety of materials. The temperature coefficient of these devices is specified in terms of ohms, per ohms per °C.*

In most applications, linearization is not required. Table 6 shows the temperature versus resistance of a 100 Ω platinum RTD. With a 100Ω PRTD, the change in resistance from 0°C to 100°C changes resistance by:

$$\Delta R = (Thermal\ Response)\ x\ R_0\ x\ \Delta t$$

$$\Delta R = 0.00038\Omega/\Omega/^\circ C\ x\ 100\Omega\ x\ 100^\circ C$$

$$\Delta R = 38.5\Omega$$

The accuracy of the PRTD over its temperature range is also shown in terms of Δ°C from ideal.

Of the temperature sensors discussed in this application note, the RTD is the most linear with only two coefficients in the linearization equation,

$$R_t = R_0(1 + At + Bt^2)$$

for temperatures 0 °C to 859 °C

$$R_t = R_0(1 + At + Bt^2) + C(t - 100t^3)$$

for temperatures -200°C to 0°C
where
R_t is the resistance of the RTD at measurement temperature,
t is the temperature being measured,
R_0 is the magnitude of the RTD at 0°C,
A, B and C are calibration coefficients derived from experimentation.

These equations are solved after five iterations making it possible to resolve to ±0.001°C of accuracy.

RTD Error Analysis

Beyond the initial element errors shown in Table 6 there are other sources of error that effect the overall accuracy of the temperature sensor. The introduction of defects into the mechanical integrity of the part such as bending the wires, shock due to rough handling, constriction of the packaging that leads to stress during thermal expansion, and vibration can have a long term effect on the repeatability of the sensor.

Although the mechanical stresses can effect long term stability, the electrical design used to condition, gain and digitize the RTD output can also effect the overall accuracy. One of these sources of errors is the self heating of the RTD element that results from the required current excitation. A current excitation is used to convert the resistance of the RTD into a voltage. It is desirable to have a high excitation current through the resistive sensing element in order to keep the output voltage above the system noise levels. A negative side to this design approach is that the element will self-heat as a result of the higher current. The combination of current and resistance create power and in turn the by-product of heat. The heat generated by the power dissipation of the element artificially increases the resistance of the RTD.

The error contribution of the heat generated by the element's power dissipation is easily calculated given the package thermal resistance ($\theta_{PACKAGE}$), the magnitude of the current excitation and the value of the RTD resistance (R_{RTD}).

Temperature (°C)	Typical Absolute Resistive Value (Ω)	Deviation in Ω	Deviation in °C
-200	23.0	± 0.56	± 1.3
-100	61.5	± 0.32	± 0.8
0	100.0	± 0.12	± 0.3
100	138.5	± 0.30	± 0.8
200	177.0	± 0.48	± 1.3
300	215.5	± 0.64	± 1.8
400	254.0	± 0.79	± 2.3
500	292.5	± 0.93	± 2.8
600	331.0	± 1.06	± 3.3
700	369.5	± 1.17	± 3.8
800	408.0	± 1.28	± 4.3

TABLE 6: *OMEGA Platinum Resistance Elements Allowable Deviation from Ideal Values for a 100Ω Sensor. The PRTD in this illustration is manufactured to have a thermal response of 0.00385Ω/Ω / °C (IEC 751) near 0°C, Class B.*

For example, if the package thermal resistance is 50°C/W, the RTD's nominal resistance is 250Ω, and the element is excited with a 5mA current source, the artificial increase in temperature (Δ °C) as a result of self heating is:

$$\Delta\,^{\circ}C = I^2 R_{RTD} * \theta_{PACKAGE}$$

$$\Delta\,^{\circ}C = (5mA)^2 \text{ x } 250\Omega \text{ x } 50^{\circ}C/Watt$$

$$\Delta\,^{\circ}C = 0.3125\,^{\circ}C$$

This example illustrates the importance of keeping the magnitude of current excitation as low as possible, preferably less than 1mA.

A second source of error resulting from the electrical design comes from the lead wires to and from the sensing element. The technique used to connect the RTD to the rest of the circuit can be a critical issue. Three possible wire configurations can be used when connecting the element to the remainder of the circuit. In Figure 3a. the 2-wire configuration is by far the least expensive, however, the current that is used to excite the RTD element flows through the wires as well are the resistive element. A portion of the wires are exposed to the same temperatures as the RTD. The effects of the wire resistance change with temperature can become a critical issue.

For example, if the lead wire is constructed of 5 gage copper leads that are 50 meters long (with a wire resistance of 1.028Ω/km), the contribution of both wires increases the RTD resistance by 0.1028Ω. This translates into a temperature measurement error of 0.26°C for a 100Ω @ 0°C RTD. This error contributes to the non-linearity of the overall measurement. The least accurate of configurations shown in Figure 3 is the 2-wire. Circuits can be configured to effectively use the 3-wire and 4-wire configuration to remove the error contribution of the lead wires completely.

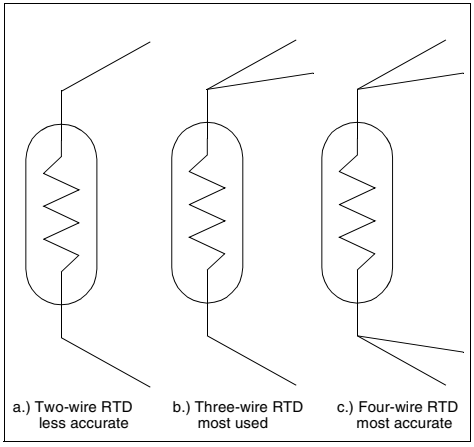


FIGURE 3: *RTD elements are available in two-wire, three-wire or four-wire configurations. Two-wire RTDs are the least accurate because the contribution of the wire resistance and wire resistance drift to the measurement. With four-wire RTDs, this error can be eliminated by using force and sense techniques in the circuit design.*

GET THE GREAT ACCURACY OF THE THERMISTOR

If accuracy is a high priority, the thermistor should be the temperature sensor of choice. Thermistors are available in two varieties, NTC and PTC. The NTC (negative temperature coefficient) thermistor is constructed of ceramics composed of oxides of transition metals (manganese, cobalt, copper, and nickel). With a current excitation the NTC has a negative temperature coefficient that is very repeatable and fairly linear. These temperature dependent semiconductor resistors operate over a range for –100°C to 450°C. Combined with the proper packaging, they have a continuous change of resistance over temperature. This resistive change versus temperature is larger than the RTD (see Figure 4), consequently the thermistor is systematically more sensitive.

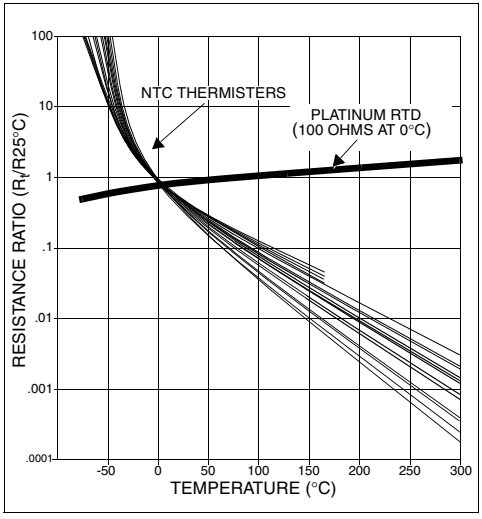


FIGURE 4: *The temperature response versus resistance of the NTC thermistor and the RTD.*

The temperature characteristics of a typical NTC thermistor along with a 100Ω RTD is shown in Figure 4. In this figure, the difference between the temperature coefficients of these two sensors is noticeable. The thermistor has a negative temperature coefficient as expected and the absolute value of the sensor changes by 10,000 times over its usable temperature range. In contrast, the RTD shown has a positive temperature coefficient and only changes by four times over is usable temperature range.This higher sensitivity of the thermistor makes it attractive in terms of accuracy in measurements.

The Thermistor is less linear than the RTD in that it requires a 3rd order polynomial for precise temperature corrections. The linearity equations for the Thermistor are:

$$In\ R_T = B_0 + \frac{B_1}{t} + \frac{B_2}{t^2} + \frac{B_3}{t^3}$$

over the entire temperature range

where

B_X are the material constants of the thermistor

This linearization formula can resolve to a total measurement uncertainty of ±0.005°C. However, it is tedious when implemented in the microcontroller. Alternatively, look-up tables can be generated to serve the same purpose with slightly less accuracy.

Thermistor Error Analysis

Although the NTC thermistor has the capability of being more accurate than the RTD temperature sensor, the two sensors have many things in common. They are both temperature sensitive resistors.

When using the thermistor, an error due to overheating is easily created. As a matter of fact, more care is required when designing the excitation of the thermistor because the thermistor resistive values are usually higher than the RTD. Take for example, a package thermal resistance of 10°C/W (bead diameter of 14mils), a nominal Thermistor resistance is 10kΩ @ 25°C with the Thermistor excitation of 5mA. The artificial increase in temperature (Δ°C) as a result of self heating is:

$$\Delta\ ^\circ C = I_2 R_{THERMISTOR}\ x\ \theta_{PACKAGE}$$

$$\Delta\ ^\circ C = (5mA)^2 x\ 10k\Omega\ x\ 10\ ^\circ C/Watt$$

$$\Delta\ ^\circ C = 2.5\ ^\circ C$$

With temperature changes of this nature, the measurement is obviously inaccurate, but also the thermal coefficient of the thermistor material delays the full effect of the problem for several seconds as the package material stabilizes. To complicate this thermal effect further, the thermal heating of the thermistor decreases the thermistor resistance (instead of the increase seen with the RTD). Since the thermistor has a negative resistive coefficient, the overheating effect reverses as the thermistor resistance becomes less than the voltage across the thermistor divided by the excitation current.This phenomena is not easily overcome with software calibration and should be avoided.

The PTC thermistor has a positive temperature coefficient and is constructed from barium titanate. The sensitivity of the PTC is considerably higher than the sensitivity of the NTC thermistor and should be used when a specific temperature range is of interest (-25 to 150°C). Over the lower portion of the resistance versus temperature curve the thermistor resistance if fairly constant. At higher temperatures the material passes through a threshold temperature (between 80°C and 140°C, dependent on chemical composition of the ceramic) where the resistance versus temperature characteristics change dramatically (Figure 5).

At this point, increases in temperature cause a rise in the PTC's resistance and the PTC resistive / temperature characteristics become very steep.

A second type of PTC thermistor is known as the Silistor. This device is constructed of a thermally sensitive silicon material and also has a positive temperature coefficient (-60°C to 150°C) that is linear over the entire operating range.

Both of the thermal characteristics of the PTC type thermistors are shown in Figure 5.

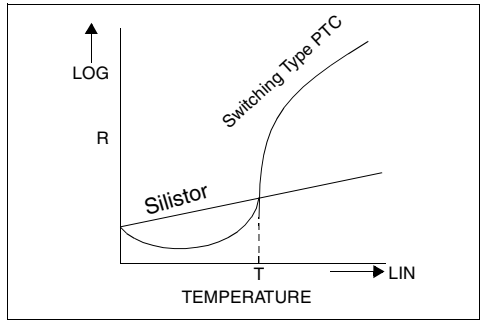


FIGURE 5: *PTC thermistor and silistor resistance versus temperature response.*

SELECT THE EASY TO USE INTEGRATED SILICON TEMPERATURE SENSOR

The integrated circuit temperature sensors offer another alternative to solving temperature measurement problems. The advantages of integrated circuit silicon temperature sensors include, user friendly output formats and ease of installation in the PCB assembly environment.

Since the silicon temperature sensor is an integrated circuit, integrated circuit designs can be easily implement on the same silicon as the sensor. This advantage allows the placement of the most challenging portions of the sensor signal conditioning path to be included in the IC chip. Consequently, the output signals from the sensor, such as large signal voltages, current, or digital words, are easily interfaced with other elements of the circuit. As a matter of fact, some integrated silicon sensors include extensive signal processing circuitry, providing a digital I/O interface for the microcontroller.

On the other hand, the accuracy and temperature range of this sensor does not match the other types of sensors discussed in this application note. A temperature sensor IC can operate over a nominal temperature range of –55 to 150 °C. Some devices go beyond this range, while others operate over a narrower range.

CHOOSE THE RIGHT TEMPERATURE SENSOR

Of the temperature sensors on the market today, the thermocouple, RTD, Thermistor, and Integrated Silicon Sensors are continuing to dominate. The thermocouple is most appropriate for higher temperature sensing, while the RTD is best suited for lower temperatures were good linearity is desirable. The Thermistor is typically used for applications with smaller temperature ranges, but it offers greater accuracy than the thermocouple or the RTD.

All four of the sensors mentioned in this application note have the capability of providing good, accurate, and reliable performance, making the final sensor selection appear somewhat trivial. However, once the temperature sensor has been selected, the next step is to design the analog and digital signal conditioning circuit. The design of this circuit will determine the actual performance that is finally achieved.

Several application notes can be found in the Microchip's library that elaborate on these circuits. Each of these application notes will present circuit alternatives that take into account simplicity, accuracy and cost.

REFERENCES

Baker, Bonnie, "Low Power Temperature Sensing with Precision Converters", *Sensors*, (February 1997) p 38.

Baker, Bonnie, "Precision Temperature Sensing with RTD Circuits", AN687, Microchip Technology Inc. (1998).

Baker, Bonnie, "Single Supply Temperature Sensing with Thermocouples", AN684, Microchip Technology Inc. (1998).

Baker, Bonnie, "Thermistors in Single Supply Temperature Sensing Circuits", AN685, Microchip Technology Inc. (1998).

Klopfenstein, Rex, "Software Linearization of a Thermocouple", *Sensors*, (December 1997) p 40.

Product Book, *Thermometrics, Inc.* (1997).

Schraff, Fred. "Thermocouple Basics" *Measurement & Control*, (June 1996) p 126.

Sulciner, James, "Understanding and Using PRTD Technology, Part 1: History, Principles and Designs", *Sensors*, (August 1996).

http://www.omega.com/techref/

NOTES:

AN682

Using Single Supply Operational Amplifiers in Embedded Systems

Author: *Bonnie Baker*
Microchip Technology Inc.

INTRODUCTION

Beyond the primitive transistor, the operational amplifier is the most basic building block for analog applications. Fundamental functions such as gain, load isolation, signal inversion, level shifting, adding and/or subtracting signals are easily implemented with this building block. More complex circuits can also be implemented, such as the instrumentation amplifier, a current to voltage converter, and filters, to name only a few. Regardless of the level of complexity of the operational amplifier circuit, knowing the fundamental operation and behavior of this building block will save a considerable amount of upfront design time.

Formal classes on this subject can be very comprehensive and useful. However, many times they fall short in terms of experience or common sense. For instance, a common mistake that is made when designing with operational amplifiers is to neglect to include the bypass capacitors in the circuit. Operational amplifier theory often overlooks this practical detail. If the bypass capacitor is missing, the amplifier circuit could oscillate at a frequency that "theoretically" doesn't make sense. If text book solutions are used, this is a difficult problem to solve.

This application note is divided into three sections. The first section will list the fundamental amplifier applications with the design equations included. These amplifier circuits where selected with embedded system integration in mind.

The second section will use these fundamental circuits to build useful amplifier functions in embedded control applications.

The third section will identify the most common single supply operational amplifier (op amp) circuit design mistakes. This list of mistakes have been gathered over many years of trouble shooting circuits with numerous designers in the industry. The most common design pitfalls can easily be avoided if the check list from this short tutorial is used.

FUNDAMENTAL OPERATIONAL AMPLIFIER CIRCUITS

The op amp is the analog building block that is analogous to the digital gate. By using the op amp in the design, circuits can be configured to modify the signal in the same fundamental way that the inverter, AND, and OR gates do in digital circuits. In this section, fundamental building blocks such as the voltage follower, non-inverting gain and inverting gain circuits will be discussed. This will be followed by a rail splitter, difference amplifier, summing amplifier and current to voltage converter.

Voltage Follower Amplifier

Starting with the most basic op amp circuit, the buffer amplifier (shown in Figure 1) is used to drive heavy loads, solve impedance matching problems, or isolate high power circuits from sensitive, precise circuitry.

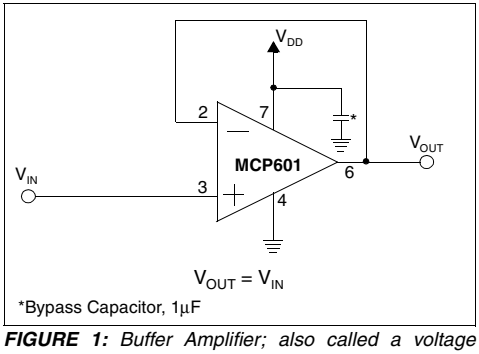


FIGURE 1: *Buffer Amplifier; also called a voltage follower.*

The buffer amplifier, shown in Figure 1, can be implemented with any single supply, unity gain stable amplifier. In this circuit as with all amplifier circuits, the op amp must be bypassed with a capacitor. For single supply amplifiers that operate in bandwidths from DC to megahertz, a 1µF capacitor is usually appropriate. Sometimes a smaller bypass capacitor is required for amplifiers that have bandwidths up to the 10s of megahertz. In these cases a 0.1µF capacitor would be appropriate. If the op amp does not have a bypass capacitor or the wrong value is selected, it may oscillate.

The analog gain of the circuit in Figure 1 is +1 V/V. Notice that this circuit has positive overall gain but the feedback loop is tied from the output of the amplifier to the inverting input. An all too common error is to assume that an op amp circuit that has a positive gain requires positive feedback. If positive feedback is used, the amplifier will most likely drive to either rail at the output.

This amplifier circuit will give good linear performance across the bandwidth of the amplifier. The only restrictions on the signal will occur as a result of a violation of the input common-mode and output swing limits. These limitations will be discussed in the third section of this application note ("Amplifier Design Pitfalls").

If this circuit is used to drive heavy loads, the amplifier that is actually selected must be specified to provide the required output currents. Another application where this circuit may be used is to drive capacitive loads. Not every amplifier is capable of driving capacitors without becoming unstable. If an amplifier can drive capacitive loads, the product data sheet will highlight this feature. However, if an amplifier can't drive capacitive loads, the product data sheets will not explicitly say.

Another use for the buffer amplifier is to solve impedance matching problems. This would be applicable in a circuit where the analog signal source has a relatively high impedance as compared to the impedance of the following circuitry. If this occurs, there will be a voltage loss with the signal as a consequence of the voltage divider between the source's impedance and the following circuitry's impedance. The buffer amplifier is a perfect solution to the problem. The input impedance of the non-inverting input of an amplifier can be as high as 10^{13} Ω for CMOS amplifiers. In addition, the output impedance of this amplifier configuration is usually less than 10 Ω.

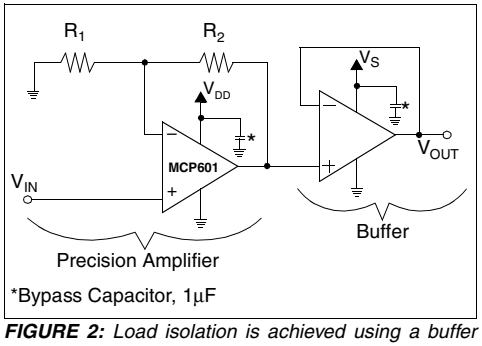


FIGURE 2: *Load isolation is achieved using a buffer amplifier.*

Yet another use of this configuration is to separate a heat source from sensitive precision circuitry, as shown in Figure 2. Imagine that the input circuitry to this buffer amplifier is amplifying a 100µV signal. This type of amplification is difficult to do with any level of accuracy in the best of situations. This precision measurement can easily be disrupted by changing the output current drive of the device that is doing the amplification work. An increase in current drive will cause self heating of the chip which will induce an offset change. An analog buffer can be used to perform the function of driving heavy loads while the front end circuitry can be used to make precision measurements.

Gaining Analog Signals

The buffer solves a lot of analog signal problems, however, there are instances in circuits where a signal needs to be gained. Two fundamental types of amplifier circuits can be used. With the first type, the signal is not inverted as shown in Figure 3. This type of circuit is useful in single supply[1] amplifier applications where negative voltages are usually not possible.

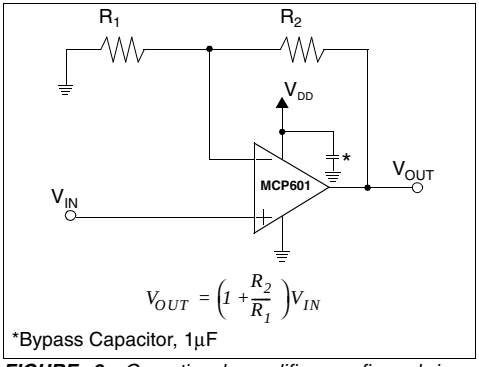


FIGURE 3: *Operational amplifier configured in a non-inverting gain circuit.*

The input signal to this circuit is presented to the high impedance, non-inverting input of the op amp. The gain that the amplifier circuit applies to the signal is equal to:

$$V_{OUT} = \left(1 + \frac{R_2}{R_1}\right)V_{IN}$$

Typical values for these resistors in single supply circuits are above 2kΩ for R_2. The resistor, R_1, restrictions are dependent on the amount of gain desired versus the amount of amplifier noise and input offset voltage as specified in the product data sheet of the op amp.

Once again, this circuit has some restrictions in terms of the input and output range. The non-inverting input is restricted by the common-mode range of the amplifier. The output swing of the amplifier is also restricted as stated in the product data sheet of the individual amplifier. Most typically, the larger signal at the output of the amplifier causes more signal clipping errors than the smaller signal at the input. If undesirable clipping occurs at the output of the amplifier, the gain should be reduced.

1. For this discussion, single supply implies that the negative supply pin of the operational amplifier is tied to ground and the positive supply pin is tied to +5V. All discussion in this application note can be extrapolated to other supply voltages where the single supply exceeds 5V or dual supplies are used.

An inverting amplifier configuration is shown in Figure 4. With this circuit, the signal at the input resistor, R_1, is gained and inverted to the output of the amplifier. The gain equation for this circuit is:

$$V_{OUT} = -\left(\frac{R_2}{R_1}\right)V_{IN} + \left(1 + \frac{R_2}{R_1}\right)V_{BIAS}$$

The ranges for R_1 and R_2 are the same as in the non-inverting circuit shown in Figure 3.

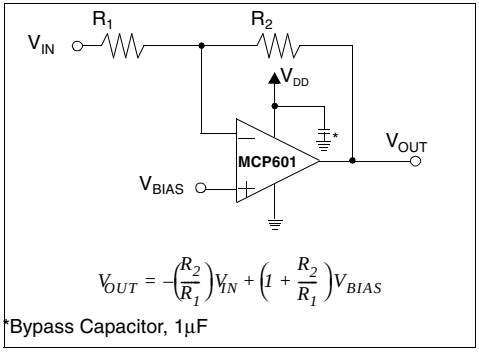


FIGURE 4: *Operational amplifier configured in an inverting gain circuit. In single supply environments a V_{BIAS} is required to insure the output stays above ground.*

In single supply applications, this circuit can easily be misused. For example, let R_2 equal 10kΩ, R_1 equal 1kΩ, VBIAS equal 0V, and the voltage at the input resistor, R_1, equal to 100mV. With this configuration, the output voltage would be –1V. This would violate the output swing range of the operational amplifier. In reality, the output of the amplifier would go as near to ground as possible.

The inclusion of a DC voltage at VBIAS in this circuit solves this problem. In the previous example, a voltage of 225mV applied to VBIAS would level shift the output signal up 2.475V. This would make the output signal equal (2.475V – 1V) or 1.475V at the output of the amplifier. Typically, the average output voltage should be designed to be equal to VDD/2.

Single Supply Circuits and Supply Splitters

As was shown in the inverting gain circuit (Figure 4), single supply circuits often need a level shift to keep the signal between negative (usually ground) and positive supply pins. This level shift can be designed with a single amplifier and a combination of resistors and capacitors as shown in Figure 5. Many times a simple buffer amplifier without compensation capacitors will accomplish this task. In other cases the level shift circuit will see dynamic or transient load changes, like the reference to an Analog-to-Digital (A/D) converter. In these applications, the level shift circuit must hold its voltage constant. If it does change, a conversion error might be observed.

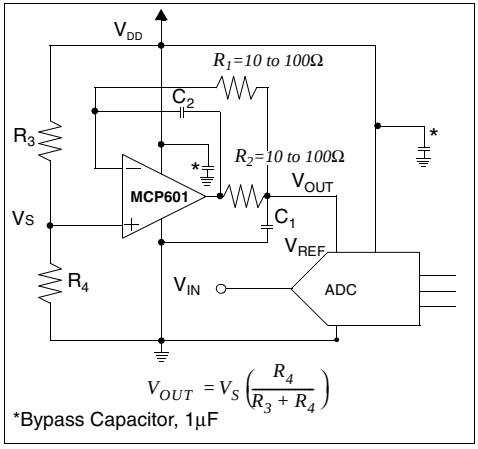


FIGURE 5: *A supply splitter is constructed using one operational amplifier. This type of function is particularly useful in single supply circuits.*

A solid level shift voltage can easily be implemented using a voltage divider (R_3 and R_4) or a reference voltage source buffered by the amplifier. The transfer function for this circuit is:

$$V_{OUT} = V_S\left(\frac{R_4}{R_3 + R_4}\right)$$

The circuit in Figure 5 has an elaborate compensation scheme to allow for the heavy capacitive load, C_1. The benefit of this big capacitor is that it presents a very low AC resistance to the reference pin of the A/D converter. In the AC domain, the capacitor serves as a charge reservoir that absorbs any momentary current surges which are characteristic of sampling A/D converter reference pins.

The Difference Amplifier

The difference amplifier combines the non-inverting amplifier and inverting amplifier circuits of Figure 3 and Figure 4 into a signal block that subtracts two signals. The implementation of this circuit is shown in Figure 6.

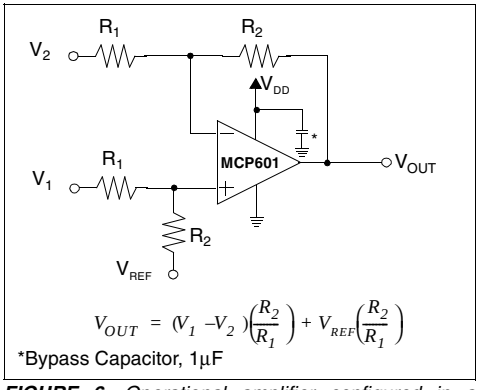


FIGURE 6: *Operational amplifier configured in a difference amplifier circuit.*

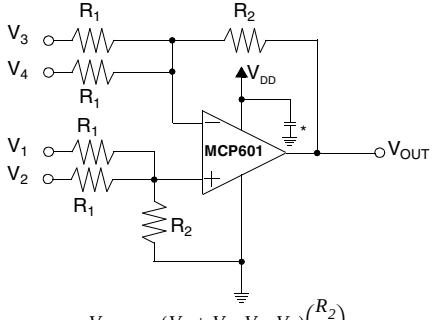
The transfer function for this amplifier circuit is:

$$V_{OUT} = (V_1 - V_2)\left(\frac{R_2}{R_1}\right) + V_{REF}\left(\frac{R_2}{R_1}\right)$$

This circuit configuration will reliably take the difference of two signals as long as the signal source impedances are low. If the signal source impedances are high with respect to R_1, there will be a signal loss due to the voltage divider action between the source and the input resistors to the difference amplifier. Additionally, errors can occur if the two signal source impedances are mismatched. With this circuit it is possible to have gains equal to or higher than one.

Summing Amplifier

Summing amplifiers are used when multiple signals need to be combined by addition or subtraction. Since the difference amplifier can only process two signals, it is a subset of the summing amplifier.



$$V_{OUT} = (V_1 + V_2 - V_3 - V_4)\left(\frac{R_2}{R_1}\right)$$

FIGURE 7: *Operational amplifier configured in a summing amplifier circuit.*

The transfer function of this circuit is:

$$V_{OUT} = (V_1 + V_2 - V_3 - V_4)\left(\frac{R_2}{R_1}\right)$$

Any number of inputs can be used on either the inverting or non-inverting input sides as long as there are an equal number of both with equivalent resistors.

Current to Voltage Conversion

An operational amplifier can be used to easily convert the signal from a sensor that produces an output current, such as a photodetector, into a voltage. This is implemented with a single resistor and an optional capacitor in the feedback loop of the amplifier as shown in Figure 8.

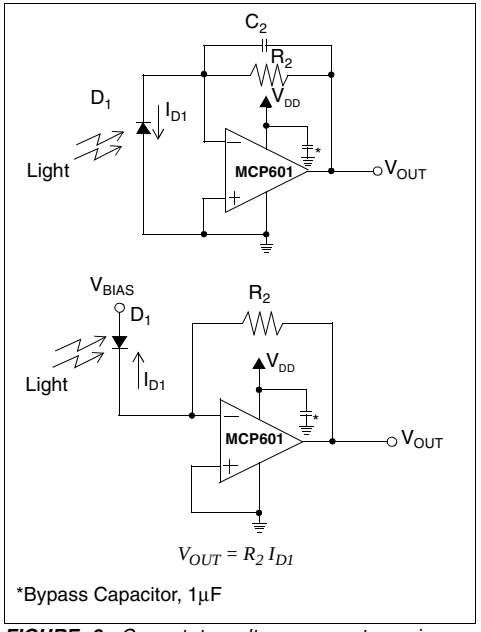


FIGURE 8: *Current to voltage converter using an amplifier and one resistor. The top light scanning circuit is appropriate for precision applications. The bottom circuit is appropriate for high speed applications.*

As light impinges on the photo diode, charge is generated, causing a current to flow in the reverse bias direction of the photodetector. If a CMOS op amp is used, the high input impedance of the op amp causes the current from the detector (I_{D1}) to go through the path of lower resistance, R_2. Additionally, the op amp input bias current error is low because it is CMOS (typically <200 pA). The non-inverting input of the op amp is referenced to ground which keeps the entire circuit biased to ground. These two circuits will only work if the common mode range of the amplifier includes zero.

Two circuits are shown in Figure 8. The top circuit is designed to provide precision sensing from the photodetector. In this circuit the voltage across the detector is nearly zero and equal to the offset voltage of the amplifier. With this configuration, current that appears across the resistor, R_2, is primarily a result of the light excitation on the photodetector.

The photosensing circuit on the bottom of Figure 8 is designed for higher speed sensing. This is done by reverse biasing the photodetector, which reduces the parasitic capacitance of the diode. There is more leakage through the diode which causes a higher DC error.

USING THE FUNDAMENTALS

Instrumentation Amplifier

Instrumentation amplifiers are found in a large variety of applications from medical instrumentation to process control. The instrumentation amplifier is similar to the difference amplifier in that it subtracts one analog signal from another, but it differs in terms of the quality of the input stage. A classic, three op amp instrumentation amplifier is illustrated in Figure 9.

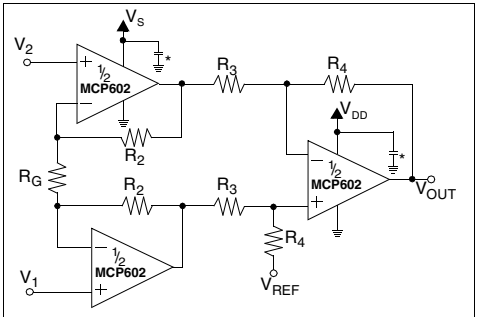


FIGURE 9: *An instrumentation amplifier can be designed using three amplifiers. The input operational amplifiers provide signal gain. The output operational amplifier converts the signal from two inputs to a single ended output with a difference amplifier.*

With this circuit the two input signals are presented to the high impedance non-inverting inputs of the amplifiers. This is a distinct advantage over the difference amplifier configuration when source impedances are high or mismatched. The first stage also gains the two incoming signals. This gain is simply adjusted with one resistor, R_G.

Following the first stage of this circuit is a difference amplifier. The function of this portion of the circuit is to reject the common mode voltage of the two input signals as well as differentiate them. The source impedances of the signals into the input of the difference amplifier are low, equivalent and well controlled.

The reference voltage of the difference stage of this instrumentation amplifier is capable of spanning a wide range. Most typically this node is referenced to half of the supply voltage in a signal supply application. A supply splitter such as the circuit in Figure 5 can be used for this purpose. The transfer function of this circuit is:

$$V_{OUT} = (V_1 - V_2)\left(1 + \frac{2R_2}{R_G}\right)\left(\frac{R_4}{R_3}\right) + V_{REF}\left(\frac{R_4}{R_3}\right)$$

A second instrumentation amplifier is shown in Figure 10. In this circuit, the two amplifiers serve the functions of load isolation, and signal gain. The second amplifier also differentiates the two signals.

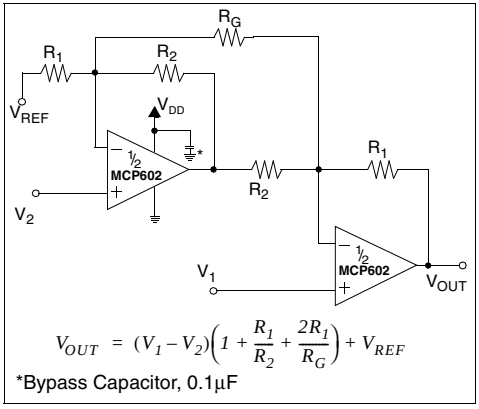


FIGURE 10: *An instrumentation amplifier can be designed using two amplifiers. This configuration is best suited for higher gains. (gain ≥ 3 V/V)*

The circuit reference voltage is supplied to the first op amp in the signal chain. Typically, this voltage is half of the supply voltage in a single supply environment.

The transfer function of this circuit is:

$$V_{OUT} = (V_1 - V_2)\left(1 + \frac{R_1}{R_2} + \frac{2R_1}{R_G}\right) + V_{REF}$$

Floating Current Source

A floating current source can come in handy when driving a variable resistance, like an Resistive Temperature Device (RTD). This particular configuration produces an appropriate 1mA source for an RTD type sensor, however, it can be tuned to any current.

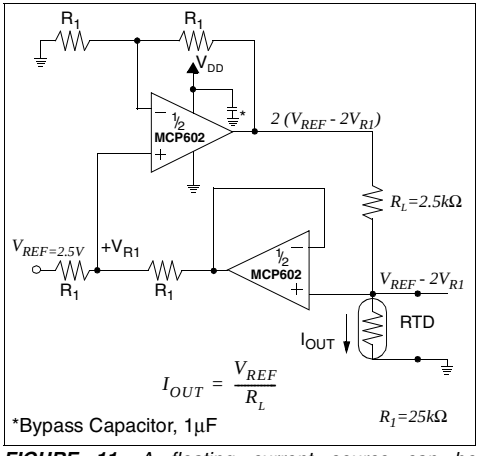


FIGURE 11: *A floating current source can be constructed using two operational amplifiers and a precision voltage reference.*

With this configuration, the voltage of V_{REF} is reduced via the first resistor, R_1, by the voltage V_{R1}. The voltage applied to the non-inverting input of the top op amp is $V_{REF} - V_{R1}$. This voltage is gained to the amplifier's output by two to equal $2(V_{REF} - V_{R1})$. Meanwhile, the output for the bottom op amp is presented with the voltage $V_{REF} - 2V_{R1}$. Subtracting the voltage at the output of the top amplifier from the non-inverting input of the bottom amplifier gives $2(V_{REF} - V_{R1}) - (V_{REF} - 2V_{R1})$ which equals V_{REF}.

The transfer function of the circuit is:

$$I_{OUT} = \frac{V_{REF}}{R_L}$$

Filters

Bandpass and low pass filters are very useful in eliminating unwanted signals prior to the input of an A/D converter. The low pass filter shown in Figure 12 has two poles that can be configured for a Butterworth filter response. Butterworth filters have a flat magnitude response in the pass-band with good all-around performance.

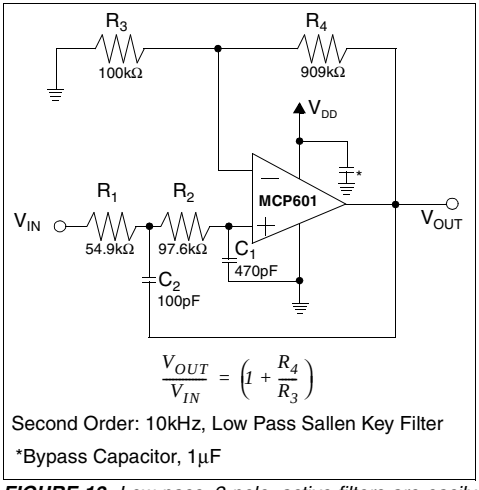


FIGURE 12: *Low pass, 2-pole, active filters are easily designed with one operational amplifier. The resistors and capacitors can be adjusted to implement other filter types, such as Bessel and Chebyshev.*

On the down side, there is some overshoot and ringing with a step response through this filter. This may or may not be an issue, depending on the application circuit requirements. The gain of this filter is adjustable with R_3 and R_4.

Notice the similarities in this gain equation and the non-inverting amplifier shown in Figure 3.

This type of filter is also referred to as an anti-aliasing filter, which is used to eliminate circuit noise in the frequency band above half of nyquist of the sampling system. In this manner, these high frequency noises, that would typically alias back into the signal path, are removed.

The DC gain of the circuit in Figure 12 is:

$$\frac{V_{OUT}}{V_{IN}} = \left(1 + \frac{R_4}{R_3}\right)$$

The bandpass filter shown in Figure 13 is configured with a zero and two poles to accommodate speech applications. The single zero high pass filter portion of this circuit is constructed with C_1 and R_1 in parallel with R_2. Notice that R_1 and R_2 also creates a supply splitter voltage at the non-inverting inputs of both of the amplifiers. This insures that both operational amplifiers operate in their linear region. The second amplifier, U_2, in conjunction with the components R_3, R_4, C_3, and C_4 set a two pole corner frequency. This filter eliminates high frequency noise that may be aliased back into the signal path.

The signal gain of this circuit is:

$$V_{OUT} = V_{IN}\left(\frac{R_3}{R_4}\right)\left(\frac{R_2}{R_1 + R_2}\right)$$

For more details about filters refer to *AN699 "Anti-aliasing Analog Filters for Data Acquisitions Systems.*

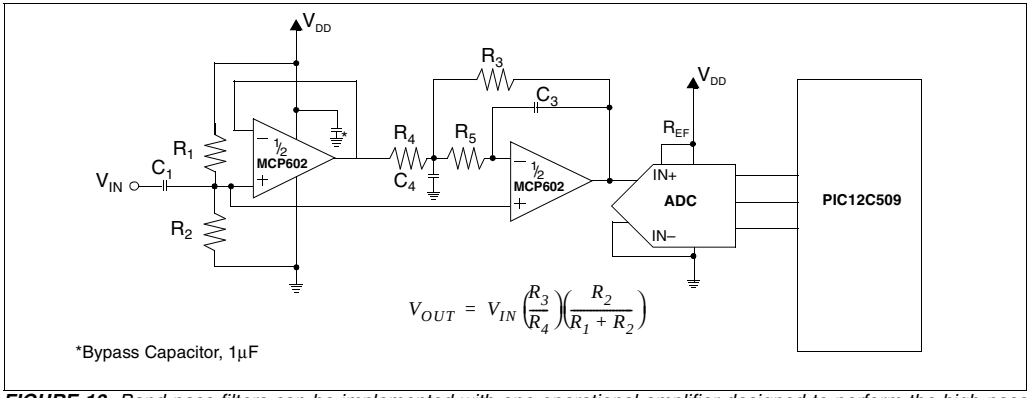


FIGURE 13: *Band pass filters can be implemented with one operational amplifier designed to perform the high pass function and a second amplifier to perform the low pass function.*

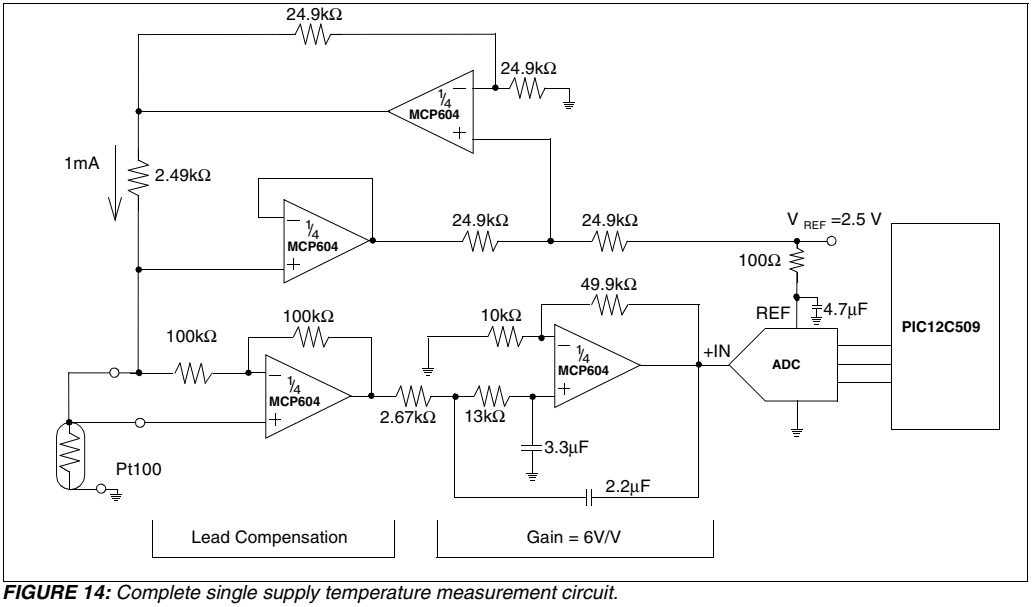


FIGURE 14: *Complete single supply temperature measurement circuit.*

Putting it Together

The circuit shown in Figure 14 utilizes four operational amplifiers along with a 12-bit A/D converter to implement a complete single supply temperature measurement circuit. The temperature sensor is an RTD which requires current excitation. The current excitation is supplied by the circuit described in Figure 11. The gain and anti-aliasing filter is implemented with the circuit shown in Figure 13.

The voltage signal from the RTD is sensed by an amplifier that is used in a combination of a non-inverting configuration and inverting configuration.

The output of this amplifier is then sent to an amplifier that is configured as a two pole, low pass filter in a gain of +6V/V. A gain of six was chosen in order to comply with the input range of A/D converter. Assuming the sampling frequency of the A/D converter is 75kHz, which is also know as the nyquist frequency, the cut-off frequency of the anti-aliasing filter (U4) is set to 10kHz. This allows plenty of bandwidth for the filter to attenuate the signal prior to 1/2 of nyquist. The A/D converter is a 12-bit Successive Approximation Register (SAR) converter that is interfaced to the PIC12C509 microcontroller.

AMPLIFIER DESIGN PITFALLS

Theoretically, the circuits within this application note work. Beyond the theory, however, there are few tips that will help get the circuit right the first time. This section, “Amplifier Design Pitfalls”, lists common problems associated with using an op amp with a power supply and an input signal on a PC Board. It is divided into four categories: General Suggestions, Input State Problems, Bandwidth Issues, and Single Supply Pitfalls. Hopefully, the most common problems with op amp implementation have been addressed within this application note, however, if you have any other inputs from experience, please e-mail your suggests to bonnie.baker@microchip.com.

In General

1. Be careful of the supply pins. Don’t make them too high per the amplifier specification sheet and don’t make them too low. High supplies will damage the part. In contrast, low supplies won’t bias the internal transistors and the amplifier won’t work or it may not operate properly.
2. Make sure the negative supply (usually ground) is in fact tied to a low impedance potential. Additionally, make sure the positive supply is the voltage you expect when it is referenced to the negative supply pin of the op amp. Placing a volt meter across the negative and positive supply pins will verify that you have the right relationship between the pins.
3. Ground can’t be trusted, especially in digital circuits. Plan your grounding scheme carefully. If the circuit has a lot of digital circuitry, consider separate ground and power planes. It is very difficult, if not impossible, to remove digital switching noise from an analog signal.
4. Decouple the amplifier power supplies with by-pass capacitors as close to the amplifier as possible. For CMOS amplifiers, a 0.1µF capacitor is usually recommended. Also decouple the power supply with a 10µF capacitor.
5. Use short lead lengths to the inputs of the amplifier. If you have a tendency to use the white perf boards for prototyping, be aware that they can cause noise and oscillation. There is a good chance that these problems won’t be a problem with the PCB implementation of the circuit.
6. Amplifiers are static sensitive! If they are damaged, they may fail immediately or exhibit a soft error (like offset voltage or input bias current changes) that will get worse over time.

Input Stage Problems

1. Know what input range is required from your amplifier. If either inputs of the amplifier go beyond the specified input range, the output will typically be driven to one of the power supply rails.
2. If you have a high gain circuit, be aware of the offset voltage of the amplifier. That offset is gained with the rest of your signal and it might dominate the results at the output of the amplifier.
3. Don’t use rail-to-rail input stage amplifiers unless it is necessary. By the way, they are only needed when a buffer amplifier circuit is used or possibly an instrumentation amplifier configuration. Any circuit with gain will drive the output of the amplifier into the rail before the input has a problem.

Do You Have the Bandwidth?

1. Account for the bandwidth of the amplifier when sending signals through the circuit. You may have designed an amplifier for a gain of 10 and find that the AC output signal is much lower than expected. If this is the case, you may have to look for an amplifier with a wider bandwidth.
2. Instability problems can usually be solved by adding a capacitor in parallel with the feedback resistor around the amplifier. This does mean typically and not always. If an amplifier circuit is unstable, a quick stability analysis will show the problem and probably the solution.

Single Supply Rail-to-Rail

1. Operational Amplifier output drivers are capable of driving a limited amount of current to the load.
2. Capacitive loading an amplifier is risky business. Make sure the amplifier is specified to handle any loads that you may have.
3. It is very rare that a single supply amplifier will truly swing rail-to-rail. In reality, the output of most of these amplifiers can only come within 50 to 200mV from each rail. Check the product data sheets of your amplifier.

REFERENCES

Sergio Franco, “Design with Operational Amplifiers and Analog Integrated Circuits”, *McGraw Hill*

Frederiksen, Thomas, “Intuitive Operational Amplifiers”, *McGraw Hill*

Williams, Jim, “Analog Circuit Design”, *Butterworth-Heinemann*

Baker, Bonnie, “Anti-aliasing Analog Filters for Data Acquisition Systems”, *AN699, Microchip Technology Inc.*

AN684

Single Supply Temperature Sensing with Thermocouples

Author: *Bonnie C. Baker*
Microchip Technology Inc.

INTRODUCTION

There is a variety of temperature sensors on the market all of which meet specific application needs. The most common sensors used to solve these application problems include the thermocouple, Resistive Temperature Detector (RTD), Thermistor, and silicon based sensors. For an overview and comparison of these sensors, refer to Microchip's *AN679, "Temperature Sensing Technologies"*.

This application note focuses on circuit solutions that use thermocouples in the design. The signal conditioning path for the thermocouple system will be discussed in this application note followed by complete application circuits.

THERMOCOUPLE OVERVIEW

Thermocouples are constructed of two dissimilar metals such as Chromel and Constantan (Type E) or Nicrosil and Nisil (Type N). The two dissimilar metals are bonded together on one end of both wires with a weld bead. This bead is exposed to the thermal environment of interest. If there is a temperature difference between the bead and the other end of the thermocouple wires, a voltage will appear between the two wires at the end where the wires are not soldered together. This voltage is commonly called the thermocouple's Electromotive Force (EMF) voltage. This EMF voltage changes with temperature without any current or voltage excitation. If the difference in temperature between the two ends (the weld bead versus the unsoldered ends) of the thermocouple changes, the EMF voltage will change as well.

There are as many varieties of thermocouples as there are metals, but some combinations work better than others. The list of thermocouples shown in Table 1 are most typically used in industry. Their behaviors have been standardized by the National Institute of Standards and Technology (NIST).The particular document from this organization that is pertinent to thermocouples is the NIST Monograph175, "Temperature-Electromotive Force Reference Functions and Tables for the Letter-Designated Thermocouple Types Based on the ITS-90". Manufacturers use these standards to qualify the thermocouples that they ship.

Thermocouple Type	Conductors	Temperature range (°C)	Seebeck Coefficient (@ 20°C)	Application Environments
E	Chromel (+) Constantan (–)	-200 to 900	62µV/°C	oxidizing, inert, vacuum
J	Iron (+) Constantan (–)	0 to 760	51µV/°C	vacuum, oxidizing reducing, inert
T	Copper (+) Constantan (–)	–200 to 371	40µV/°C	corrosive, moist, subzero
K	Chromel (+) Alumel (–)	-200 to 1260	40µV/°C	completely inert
N	Nicrosil (+) Nisil (–)	0 to 1260	27µV/°C	oxidizing
B	Platinum (30% Rhodium) (+) Platinum (6% Rhodium) (–)	0 to 1820	1µV/°C	oxidizing, inert
S	Platinum (10% Rhodium) (+) Platinum (–)	0 to 1480	7µV/°C	oxidizing, inert
R	Platinum (13% Rhodium) (+) Platinum (–)	0 to 1480	7µV/°C	oxidizing, inert

TABLE 1: *Common thermocouple types—The most common thermocouple types are shown with their standardized material and performance specifications. These thermocouple types are fully characterized by the American Society for Testing and Materials (ASTM) and specified in IST-90 units per NIST Monograph 175.*

This style of temperature sensor offers distinct advantages over other types, such as the RTD, Thermistor or Silicon sensors. As stated before, the sensor does not require any electrical excitation, such as a voltage or current source.

The price of thermocouples varies dependent on the purity of the metals, integrity of the weld bead and quality of the wire insulation. Regardless, thermocouples are relatively inexpensive as compared to other varieties of temperature sensors.

The thermocouple is one of the few sensors that can withstand hostile environments. The element is capable of maintaining its integrity over a wide temperature range as well as withstanding corrosive or toxic atmospheres. It is also resilient to rough handling. This is mostly a consequence of the heavier gages of wire used with the thermocouples construction.

The temperature ranges of the thermocouples included in Table 1 vary depending on the types of metals that are used. These ranges are also shown graphically in Figure 1. All of the voltages shown in Figure 1 are referenced to 0°C.

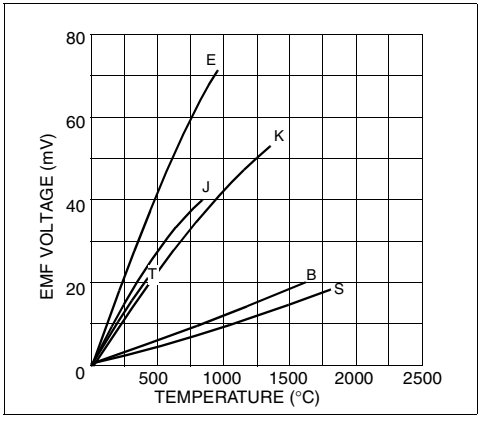


FIGURE 1: *EMF voltage of various thermocouples versus temperature*

Thermocouples produce a voltage that ranges from nano volts to tens of millivolts. This voltage is repeatable, but non-linear. Although this can be seen to a certain degree in Figure 1, Figure 2 does a better job of illustrating the non-linearity of the thermocouple. In Figure 2, the first derivative of the EMF voltage versus temperature is shown. This first derivative at a specified temperature is called the Seebeck Coefficient. The Seebeck Coefficient is a linearized estimate of the temperature drift of the thermocouple's bead over a small temperature range. Since all thermocouples are non-linear, the value of this coefficient changes with specified temperature. This coefficient is used when designing the hardware portion of the thermocouple system that senses the absolute reference temperature. The design and use of the absolute temperature reference will be discussed later in this application note.

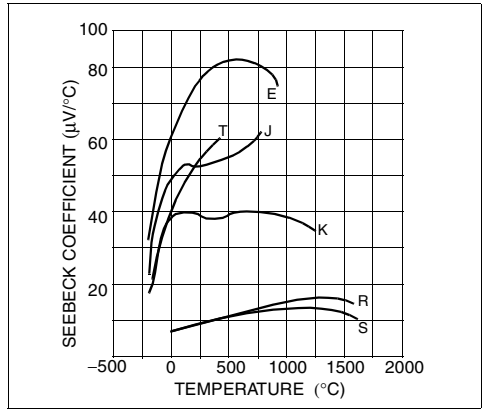


FIGURE 2: *Seebeck coefficient of various thermocouples versus temperature*

From Figure 1, it can be summized that the EMF voltage of a thermocouple is extremely small (millivolts). Additionally, Figure 2 illustrates that the change of the EMF voltage per degree C is also small (µV/°C). Consequently, the signal conditioning portion of the electronics requires an analog gain stage. In addition, the voltage that a thermocouple produces represents the temperature difference between the weld bead and the other end of the wires. If an absolute temperature measurement (as opposed to relative) is required, a portion of the thermocouple signal conditioning electronics must be dedicated to establishing a temperature reference.

A summary of the thermocouple's advantages and disadvantages are listed in Table 2.

ADVANTAGES	DISADVANTAGES
No Excitation Required	Non-Linear
Inexpensive	Needs Absolute Temperature Reference
Wide Variety of Materials	Small Voltage Output Signals
Wide Temperature Ranges	
Very Rugged	

TABLE 2: *Thermocouple Advantages and Disadvantages*

THERMOCOUPLE SIGNAL CONDITIONING PATH

The signal conditioning signal path of the thermocouple circuit is illustrated in Figure 3. The elements of the path include the thermocouple, reference temperature junction, analog gain cell, Analog-to-Digital (A/D) Converter and the linearization block. Thermocouple 1 is the thermocouple that is at the site of the temperature measurement. Thermocouple 2 and 3 are a consequence of the wires of Thermocouple 1 connecting to the copper traces of the PCB.

The remainder of this application note will be devoted to solving the reference temperature, signal gain and A/D conversion issues. Linearization issues associated with thermocouples will also be discussed.

DESIGNING THE REFERENCE TEMPERATURE SENSOR

An absolute temperature reference is required in most thermocouple applications. This is used to remove the EMF error voltage that is created by thermocouples 2 and 3 in Figure 3. The two metals of these thermocouples come from the temperature sensing element (Thermocouple 1) and the copper traces of the PCB. The isothermal block in Figure 3 is constructed so that the Thermocouples 2 and 3 are kept at the same temperature as the absolute temperature sensing device. These elements can be kept at the same temperature by keeping the circuitry in a compact area, analyzing the board for possible hot spots, and identifying thermal hot spots in the equipment enclosure. With this configuration, the known temperature of the copper junctions can be used to determine the actual temperature of the thermocouple bead.

In Figure 3, the absolute reference temperature is sensed at the isothermal block, and then subtracted from the signal path. This is a hardware implementation. Alternatively, the absolute reference temperature can be sensed and subtracted is firmware. The hardware solution can be designed to be relatively error free as will be discussed later. The firmware correction can be more accurate because of the computing power of the processor. The trade-off for this type of calibration is computing time.

The relationship between the thermocouple bead temperature and zero degrees C is published in the form of look-up tables or coefficients of polynomials in the NIST publication mentioned earlier. If the absolute temperature of thermocouple 2 and 3 (Figure 3) are known, the actual temperature at the test sight (Thermocouple 1) can be measured and then calculated.

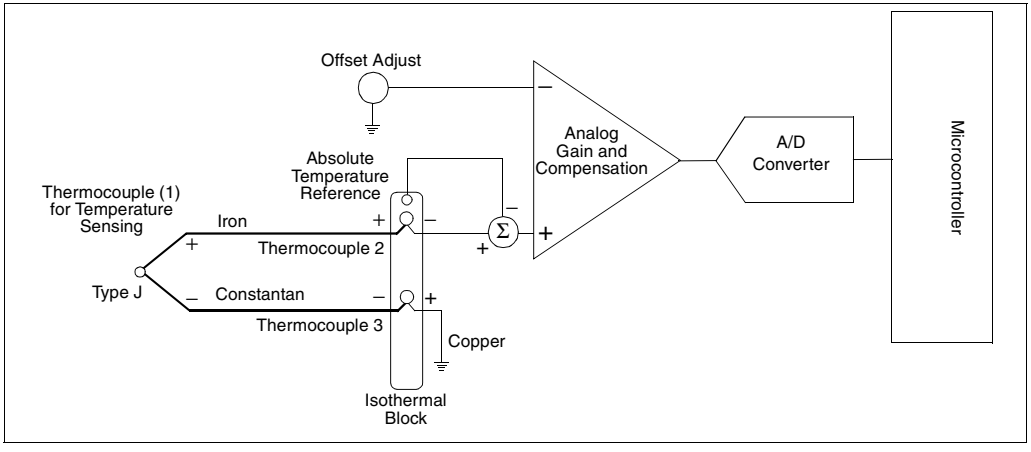


FIGURE 3: *The thermocouple signal path starts with the thermocouple which is connected to the copper traces of the PCB on the isothermal block. The signal path then continues on to a differentiating circuit that subtracts the temperature of the isothermal block from the thermocouple's temperature. After this signal is digitized, a microcontroller uses the digital word from the temperature sensing circuit for further processing.*

ERROR CORRECTION WITH HARDWARE IMPLEMENTATIONS

Many techniques can be used to sense the reference temperature on the isothermal block; five of which are discussed here. The first example uses a second thermocouple. It is used to sense ambient at the copper connection and configured to normalize the resultant voltage to an assignable temperature. As a second example, a standard diode is used to sense the absolute temperature of the isothermal block. This is done by using the negative temperature coefficient of –2.2mV/°C characteristic of the diode. Thirdly, a thermistor temperature sensor is shown as the reference temperature device. As with the diode, the thermistor has a negative temperature coefficient. The thermistor is a more challenging to use because of its non-linear tendencies, however, the price is right. Another technique discusses an RTD as the reference temperature sensor. These sensors are best suited for precision circuits. And finally, the integrated silicon temperature sensor is briefly discussed.

Using a Second Thermocouple

A second thermocouple can be used to remove the error contribution of all of the thermocouples in the circuit. A circuit that uses this technique is shown in Figure 4.

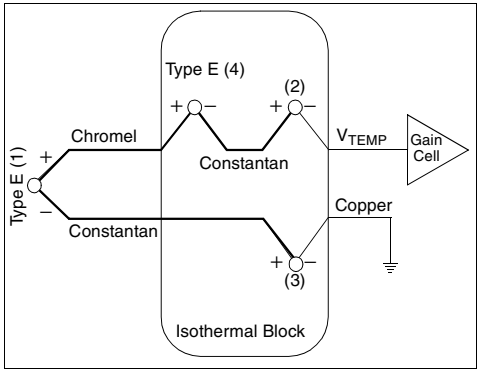


FIGURE 4: *A second temperature reference can be created by using a second thermocouple.*

In this circuit example, a Type E thermocouple is chosen to sense the unknown temperature. The Type E thermocouple is constructed of Chromel (a combination of Nickel and Chromium) on its positive side and Constantan on its negative side. A second Type E thermocouple is included in the circuit. It is positioned on the isothermal block and installed between the first thermocouple and the signal conditioning circuit.The polarity of the two Type E thermocouples is critical so that the Constantan on both of the thermocouples are connected together.

From this circuit configuration, two additional thermocouples are built, both of which are constructed with chromel and copper. These two thermocouples are opposing each other in the circuit. If both of these newly constructed thermocouples are at the same temperature, they will cancel each other's temperature induced errors.

The two remaining Type E thermocouples generate the appropriate EMF voltage that identifies the temperature at the sight of the first thermocouple.

This design technique is ideal for instances where the temperature of the isothermal block has large variations or the first derivative of voltage versus temperature of the selected thermocouple has a sharp slope (see Figure 2). Thermocouples that fit into this category in the temperature range from 0°C to 70°C are Type T and Type E.

The error calculation for this compensation scheme is:

$$V_{TEMP} = +EMF_3 + EMF_1 - EMF_4 - EMF_2$$

where

EMF_1 is the voltage drop across the Type E thermocouple at the test measurement site.

EMF_2 is the voltage drop across a Copper/Constantan thermocouple, where the copper metal is actually a PCB trace.

EMF_3 is the voltage drop across a Copper/Constantan thermocouple, where the copper metal is actually a PCB trace.

EMF_4 is the voltage drop across a Type E thermocouple on the Isothermal Block.

V_{TEMP} is the equivalent EMF voltage of a Type E thermocouple, #1, referenced to 0°C.

The temperature reference circuitry is configured to track the change in the Seebeck Coefficient fairly accurately. The dominating errors with this circuit will occur as a consequence of less than ideal performance of the Type E thermocouples, variations in the purity of the various metals, and an inconsistency in the temperature across the isothermal block.

Diode Temperature Sensing

Diodes are useful temperature sensing devices where high precision is not a requirement. Given a constant current excitation, standard diodes, such as the IN4148, have a voltage change with temperature of approximately –2.2mV/°C. These types of diodes will provide fairly linear voltage versus temperature performance. However, from part to part they may have variations in the absolute voltage drop across the diode as well as temperature drift.

This type of linearity is not well suited for thermocouples with wide variations in their Seebeck Coefficients over the temperature range of the isothermal block (referring to Figure 2). If there are wide variations with the isothermal block temperature, Type K, J, R and S

thermocouples may be best suited for the application. If the application requires more precision in terms of linearity and repeatability from part to part than an off-the-shelf diode, the MTS102, MTS103 or MTS105 from Motorola® can be substituted.

A circuit that uses a diode as an absolute temperature sensor is shown in Figure 5. A voltage reference is used in series with a resistor to excite the diode. The diode change with temperature has a negative coefficient, however, the magnitude of this change is much higher than the change of the collective thermocouple junctions on the isothermal block. This problem is solved by putting two series resistors in parallel with the diode. In this manner, the change of –2.2mV/°C of the diode is attenuated to the Seebeck Coefficient of the thermocouple on the isothermal block. The Seebeck Coefficient of the thermocouples on the isothermal block are also equal to the Seebeck Coefficient (at isothermal block temperature) of the thermocouple that is being used at the test site. Table 3 has some recommended resistance values for various thermocouple types and excitation voltages.

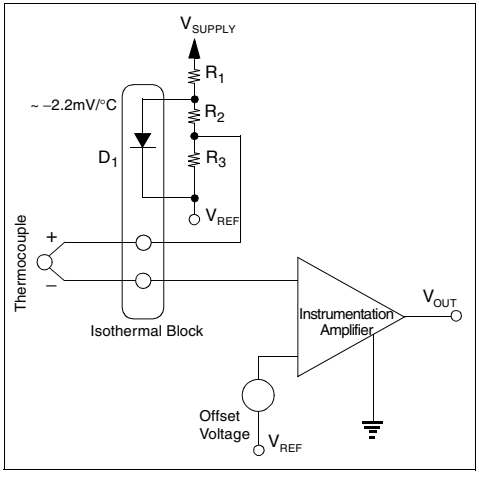


FIGURE 5: *A diode can also be used in a hardware solution to zero out the temperature errors from the isothermal block.*

This circuit appears to provide a voltage excitation for the diode. This is true, however, the ratio of the voltage excitation to the changes in voltage drop changes with temperature across the diode minimize linearity errors.

Of the three voltage references chosen in Table 3, the 10V reference provides the most linear results. It might also be noticed that changes in the reference voltage will also change the current through the diode. This being the case, a precision voltage reference is recommended for higher accuracy application requirements.

Thermocouple Type	Seebeck Coefficient (@20°C)	V_{REF} (V)	R_1 (Ω)	R_2 (Ω)	R_3 (Ω)
J	51µV/°C	4.096	9.76k	4.22k	100
J	51µV/°C	5.0	12.1k	4.22k	100
J	51µV/°C	10.0	27k	4.22k	100
K	40µV/°C	4.096	9.76k	5.36k	100
K	40µV/°C	5.0	12.1k	5.36k	100
K	40µV/°C	10.0	27k	5.36k	100
R	7µV/°C	4.096	9.76k	31.6k	100
R	7µV/°C	5.0	12.1k	31.6k	100
R	7µV/°C	10.0	27k	31.6k	100
S	7µV/°C	4.096	9.76k	31.6k	100
S	7µV/°C	5.0	12.1k	31.6k	100
S	7µV/°C	10.0	27k	31.6k	100

TABLE 3: *Recommended resistors and voltage references versus thermocouples for the circuit shown in Figure 5.*

Thermistor Circuits

Thermistors are resistive devices that have a Negative Temperature Coefficient (NTC). These inexpensive sensors are ideal for moderate precision thermocouple sensing circuits when some or all of the non-linearity of the thermistor is removed from the equation.

The NTC thermistor's non-linearity can be calibrated out with firmware or hardware techniques. The firmware techniques are more accurate, however, hardware techniques are usually more than adequate. Details on these linearity issues of thermistors are discussed in Microchip's *AN685, "Thermistors in Single Supply Temperature Sensing Circuits"*.

Figure 6 shows a thermistor in series with a equivalent resistor and voltage excitation. In this circuit, the change in voltage with temperature is ~ –25mV/°C. This temperature coefficient is too high. A resistor divider (R_1 and R_2 in Figure 6) can easily provide the required temperature coefficient dependent on the thermocouple type.

This type of voltage excitation does have fairly linear operation over a limited temperature range (0°C to 50°C). Taking advantage of this linear region reduces firmware calibration overhead significantly.

Alternatively, the NTC thermistor can be excited with a current source. Low level current sources, such as 20µA are usually recommended which minimizes self heating problems. A thermistor that is operated with current firmware excitation has a fairly non-linear output. With this type of circuit, firmware calibration would be needed. Although the firmware calibration is somewhat cumbersome, this type of excitation scheme can be more accurate.

Figure 7 compares the linearity of the thermistor with the current excitation configuration to a voltage excitation scheme shown in Figure 6.

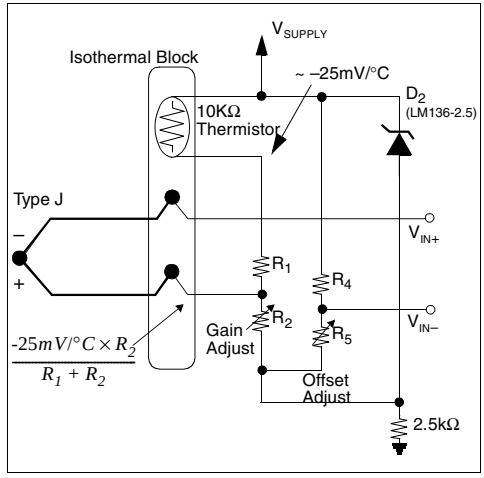


FIGURE 6: *As a third method, a thermistor is used to sense the temperature of the isothermal block. In this circuit, the isothermal block error is eliminated in hardware.*

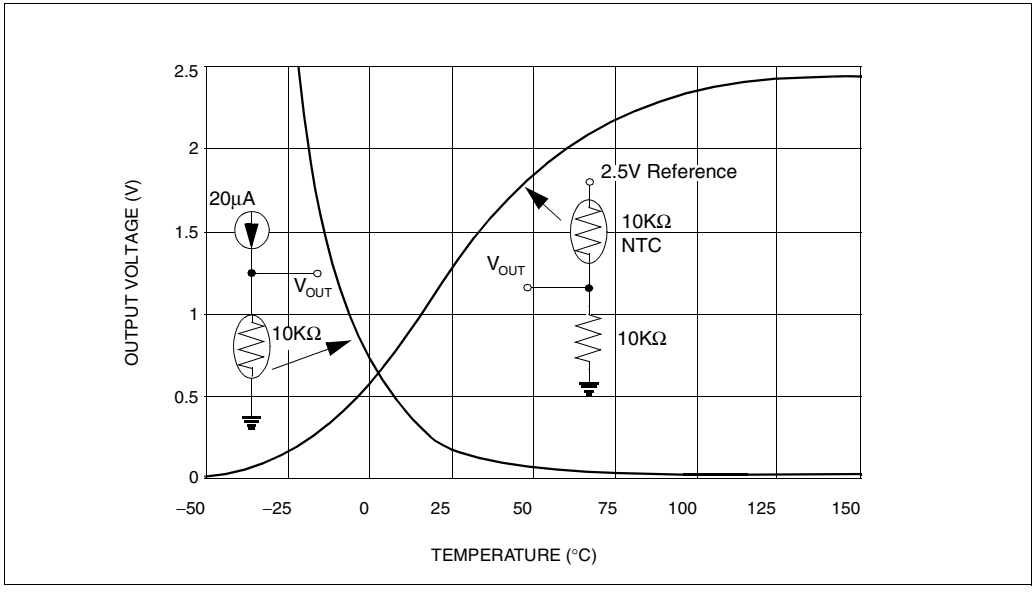


FIGURE 7: *The Thermistor in Figure 6 requires linearization. This can be accomplished by using the Thermistor in series with a standard resistor.*

RTD Sensor Circuits

Typically, an RTD would be used on the isothermal block if high precision is desired. The RTD element is nearly linear, consequently, employing linearization algorithms for the RTD is usually not required. The most effective way to get good performance from an RTD is to excite it with current. Both Figure 8 and Figure 9 show circuits that can be used for this purpose.

In Figure 8, a precision current reference is gained by the combination of R_1, R_2, J_1, U_1 and U_2. U_2 generates a 200µA precision current source. That current is pulled across R_1 forming a voltage drop for the power supply down to the non-inverting input of U_1. U_1 is used to isolate R_1 from R_2, while translating the voltage drop across R_1 to R_2. In this manner, the 200µA current from U_2 is gained by the ratio of R_1/ R_2. J_1 is used to allow the voltage at the top of the RTD element to float dependent on its resistance changes with temperature. The RTD element should be sensed differentially. The voltage across this differential output is proportional to absolute temperature.

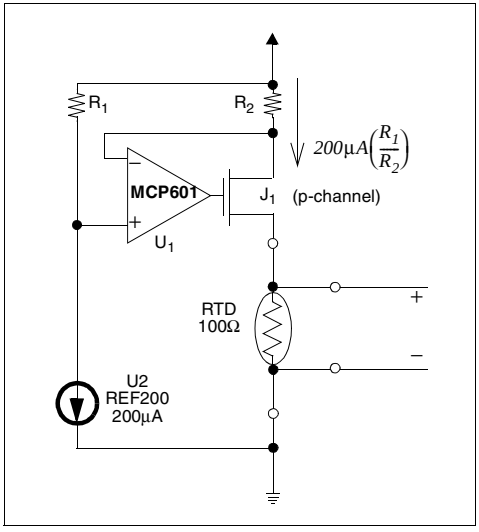


FIGURE 8: *An 4-wire RTD can be used to sense the temperature of the isothermal block. RTDs require a precision current excitation as shown here.*

In Figure 9, a voltage reference is used to generate a 1mA current source for the RTD element. The advantage of this configuration is that the voltage reference can be used elsewhere, allowing ratiometric calibration techniques in other areas of the circuit.

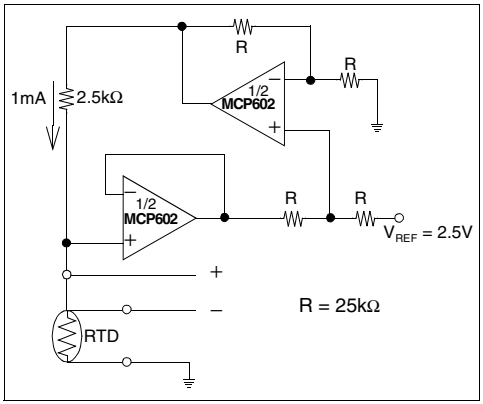


FIGURE 9: *3-wire RTD current excitation is generated with a precision voltage reference.*

The RTD sensor is best suited for situations where precision is critical. Both of the RTD circuits (Figure 8 and Figure 9) will output a voltage that is fairly linear and proportional to temperature. This voltage is then used by the microcontroller to convert the absolute temperature reading of the isothermal block back to the equivalent EMF voltage. This can be preformed by the microcontroller with a look-up table or a polynomial calculation for higher accuracy. This EMF voltage is then subtracted from the voltage measured across the sensor/isothermal block combination. In this manner, the errors from the temperature at the isothermal block are removed.

For more information about RTD circuits, refer to Microchip's *AN687, "Precision Temperature Sensing with RTD Circuits"*.

Silicon Sensor

Silicon temperature sensors are differentiated from the simple diode because of their complexity (see Figure 10). A silicon temperature sensor is an integrated circuit that uses the diode as a basic temperature sensing building block. It conditions the temperature response internally and provides a usable output such as 0 to 5V output, digital 8 or 12 bit word, or temperature-to-frequency output.

The output of this type of device is used by the processor to remove the isothermal block errors.

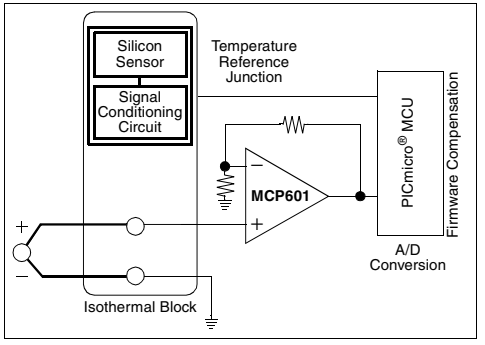


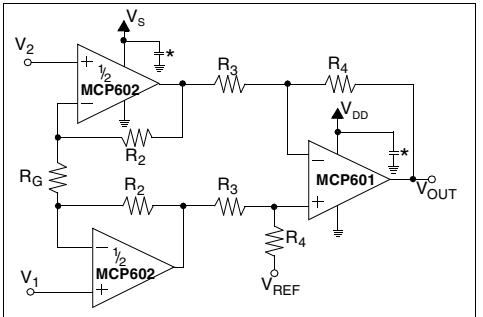
FIGURE 10: *Silicon sensors are also useful for isothermal block temperature sensing. These type of devices only sense the temperature and do not implement any error correction in hardware.*

SIGNAL CONDITIONING CIRCUITS

Once the reference temperature of the isothermal block is known, the temperature at the bead of the thermocouple can be determined. This is done by taking the EMF voltage, subtracting isothermal block errors, and determining the temperature through look-up tables or linearization equations. The EMF voltage must be digitized in order to easily perform these operations. Prior to the A/D conversion process, the low level voltage at the output of the thermocouple must be gained.

This is typically done with an instrumentation amplifier or a operational amplifier in a high gain configuration. An instrumentation amplifier uses several operational amplifiers and is configured to have a electrically equivalent differential inputs, high input impedance, potentially high gain, and good common-mode rejection. Of these four attributes, the first three are most useful for thermocouple applications.

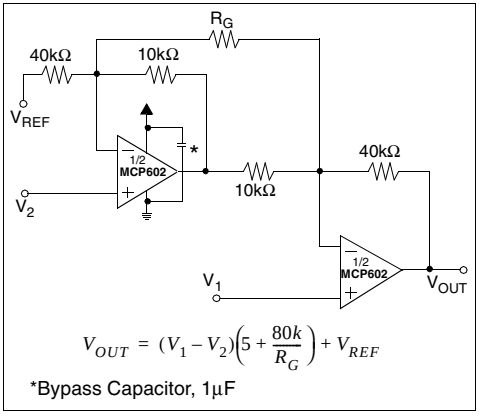
Single supply configurations of instrumentation amplifiers are shown in Figure 11 and Figure 12. In Figure 11, three operation amplifier are used along with a selection of resistors. The circuit gain in Figure 11 can be controlled with R_G.



$$V_{OUT} = (V_1 - V_2)\left(1 + \frac{2R_2}{R_G}\right)\left(\frac{R_4}{R_3}\right) + V_{REF}\left(\frac{R_4}{R_3}\right)$$

FIGURE 11: *Instrumentation amplifier using three operational amplifiers*

In Figure 12, an instrumentation amplifier is built using two amplifiers. Once again the gain is easily adjusted with R_G in the circuit.



$$V_{OUT} = (V_1 - V_2)\left(5 + \frac{80k}{R_G}\right) + V_{REF}$$

FIGURE 12: *Instrumentation amplifier using two operational amplifiers*

More details concerning the operation of Figure 11 and Figure 12 circuit configurations can be found in Microchip's *AN682, "Using Single Supply Operational Amplifiers in Embedded Systems".*

Finally, Figure 13 shows an circuit configuration using a single operational amplifier in an non-inverting gain.

These operational amplifier circuits will be used in the signal conditioning portion of the following thermocouple circuits.

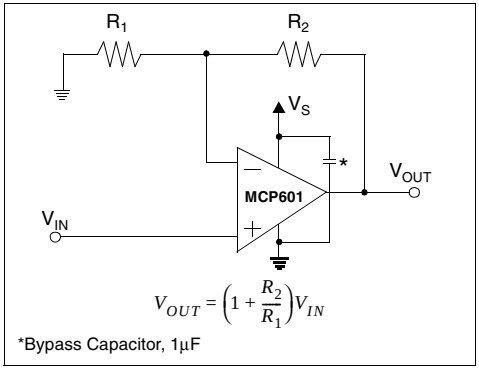


FIGURE 13: *A single operational amplifier can be configured for analog gain.*

THERMOCOUPLE CIRCUITS VERSUS ACCURACY

There are three types of thermocouple sensing systems in this section. The first circuit is designed to sense a threshold temperature. The second circuit will provide up to 8 bits of accuracy. This circuit accuracy can be improved by adding a higher resolution A/D Converter to the circuit, as shown in the third sensing system.

Threshold Temperature Sensing

A thermocouple can be used to sense threshold temperatures. This is particularly useful in industrial applications where high temperature processes need to be limited. The circuit to implement this type of function is shown in Figure 14. The threshold temperature sensing circuit in this figure combines the building blocks from Figure 4 and Figure 13.

This circuit is designed for simplicity. Consequently, all of the isothermal block error correction is performed in hardware. The Type E thermocouple is chosen for this circuit because of its high EMF voltage at high temperatures. This makes it easier to separate the real signal from background noise. Since the output of the isothermal block is single ended, the amplifier circuit in Figure 13 is used. In the event that there is a great deal of ambient or electrical noise, an instrumentation amplifier would serve this application better.

The EMF voltage of the thermocouple is calibrated across the isothermal block with a second thermocouple. This voltage is then gained by a single supply amplifier in a non-inverting configuration. The gain on the amplifier is adjustable by changing the ratio of R_2 and R_1. In this case the signal is gained by 47.3V/V using a MCP601, single supply, CMOS operational amplifier. This gain was selected to provide a 2.5V output to the amplifier for a 700°C mid-scale measurement.

The microcontroller comparator can be programmed to compare between 1.25V and 3.75V with increments of $V_{DD}/32$ (LSB size of 156.25mV). This is done by configuring the CMCON register of the PIC16C62X to CxOUT = 0 and CM<2:0> = 010. Additionally, the voltage reference to the comparator is changed in the VRCON register. The initial settings for this register is VREN = 1and VRR = 0. The processor can then cycle through the VRCON register VR<3:0> for a total of 16 different voltage reference settings for comparisons to the input signal from the MCP601 operational amplifier.

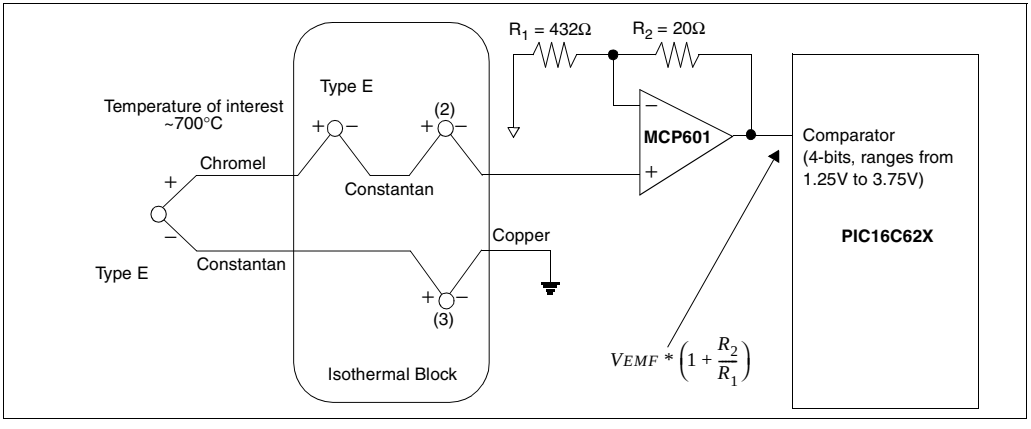


FIGURE 14: *This circuit can be used to determine temperature thresholds. With calibration, the circuit is accurate to four bits.*

°C	0	10	20	30	40	50	60	70	80	90	100
500	37.005	37.815	38.624	39.434	40.243	41.052	41.862	42.671	43.479	44.286	450.93
600	45.093	45.900	46.705	47.509	48.313	49.116	49.917	50.718	51.517	52.315	53.112
700	53.112	53.908	54.709	55.497	56.289	57.080	57.870	58.659	58.446	60.232	61.017
800	61.017	61.801	62.583	63.364	64.144	64.922	65.698	66.473	67.246	68.017	68.787
900	68.787	69.554	70.319	71.082	71.844	72.603	73.360	74.115	74.869	75.621	76.373

TABLE 4: *Type E thermocouple look-up table. All values in the tables are in millivolts.*

A look-up table for the millivolts to 500°C to 1000°C for the Type E thermocouple is provided in Table 4. The temperature at the test sight is found by dividing the output voltage of the amplifier by 47.3 and using the look-up table to estimate the actual temperature. AN566, "Implementing a Table Read" can be used in this application to program the PICmicro® microcontroller.

Measurement errors (referred to the thermocouple) in this circuit come from, the offset voltage of the operational amplifier (+/-2mV) and the comparator LSB size (+/-1.65mV). Negligible error contributions come from the look-up table resolution, resistors and power supply variations.

Given the errors above, the accuracy of the comparison in this circuit is ~ +/-35°C over a nominal temperature range of 367.7°C to 992.6°C. This error can be calibrated out. The temperature thresholds for the various settings of VR<3:0> of the VRCON register is summarized in Table 5.

This accuracy can be improved by using an amplifier with less initial offset voltage or an A/D conversion with more bits.

All of the temperature calibration work in this circuit is performed in hardware. Linearization and temperature accuracy are performed in firmware with the look-up table above.

VR<3:0>	Comparator Reference	Nominal Temperature Threshold
0000	1.25V	368.4°C
0001	1.40625V	409.8°C
0010	1.5625V	450.9°C
0011	1.71875V	491.7°C
0100	1.875V	532.6°C
0101	2.03125V	573.4°C
0110	2.1875V	614.3°C
0111	2.34375V	655.4°C
1000	2.5V	696.8°C
1001	2.65625V	738.3°C
1010	2.8125V	780.2°C
1011	2.96875V	822.3°C
1100	3.125V	864.8°C
1101	3.28125V	907.6°C
1110	3.4375V	950.9°C
1111	3.59375V	994.7°C

TABLE 5: *With a PIC16C62X controller, the comparator reference voltage is shown with the nominal temperature threshold that would be measured with the circuit in Figure 14*

Temperature Sensing up to 8-bits

An eight bit accurate thermocouple circuit is achievable by using the circuit shown in Figure 15. A Type K thermocouple is chosen for this circuit because of its stable Seebeck Coefficient between 0 and 50°C. Circuits from Figure 7 and Figure 11 are used to implement the reference temperature block as well as the signal conditioning block, respectively.

The thermistor is used as the absolute temperature sensor on the isothermal block. The combination of the thermistor and the surrounding resistors perform a first order linearization of the thermistor as discussed earlier.

The non-inverting input of the instrumentation amplifier (see Figure 11) is connected to the combination of the Type K thermocouple and the thermistor error correction circuitry. The inverting input of the instrumentation amplifier is connected to the combination of R_4 and R_5 which provide an offset adjust capability. This offset adjustment capability is not needed if the temperature sensing application starts from 0°C. However, if the temperature of interest is above a certain threshold, the offset adjust can be used to improve the dynamic range of the measurement by allowing for the full-scale range of the instrumentation amplifier and the A/D Converter to be utilized.

Assuming that the temperature range of the measurement is from 500°C to 1000°C an appropriate offset voltage at the inverting input of the instrumentation amplifier would be 43.72mV for the combination of the Type K thermocouple offset at 750°C (per Table 6) and for the thermistor absolute temperature sensing circuit at 25°C.

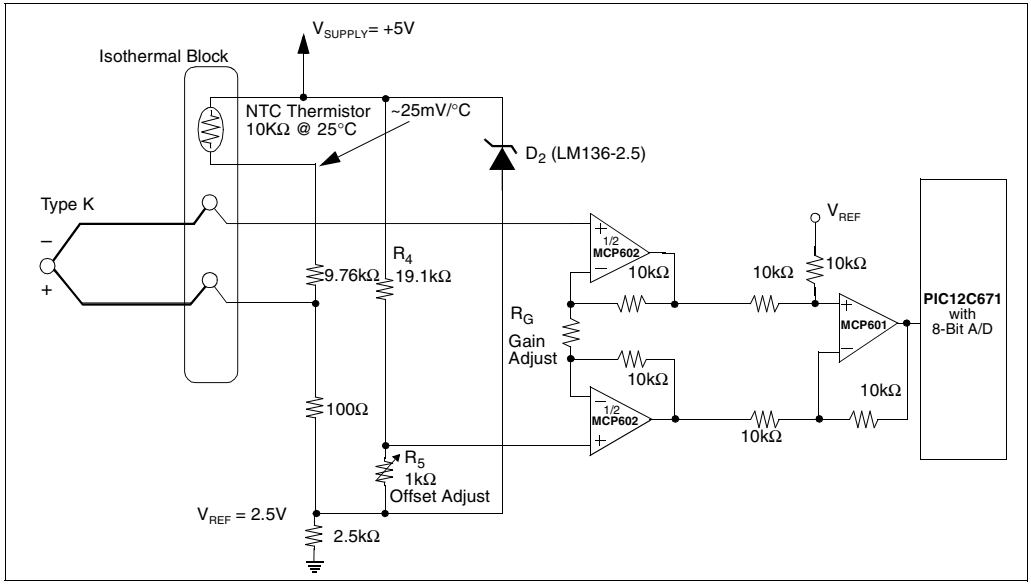


FIGURE 15: *This circuit will provide 8-bit accurate temperature sensing results using a thermocouple. In this circuit, the A/D Converter is included in the PIC12C671 microcontroller.*

°C	0	10	20	30	40	50	60	70	80	90	100
500	20.644	21.071	21.497	21.924	22.360	22.776	23.203	23.629	24.055	24.480	24.905
600	24.905	25.330	25.766	26.179	26.602	27.025	27.447	27.869	28.289	28.710	29.129
700	29.129	29.548	29.965	30.382	30.798	31.213	31.628	32.041	32.453	32.865	33.275
800	33.275	33.685	34.093	34.501	34.906	35.313	35.718	36.121	35.524	36.925	37.326
900	37.326	37.725	38.124	38.522	38.918	39.314	39.708	40.101	40.494	40.885	41.276

TABLE 6: *Type K thermocouple output voltage look-up table. All values in the table are in millivolts.*

Assuming that the offset has been minimized, the output range of the thermocouple circuit for an excursion from 500°C to 1000°C is ∆20.632mV.

The output of the instrumentation amplifier swings up to V_{DD} – 100mV. In this single supply, 5V environment, the output of the MCP601 operational amplifier will swing from 100mV to 4.9V.

The differential voltage swing at the inputs to the instrumentation amplifier is –17.41mV to +16.13mV centered around the voltage reference of 2.5V.

Given a full-scale voltage of 33.54mV from the temperature sensing circuit, the instrumentation amplifier can be configured for a gain of 137.85V/V. This gain can easily be implemented by making R_G equal to 147Ω. This circuit is not restricted to 8-bits of accuracy. An external A/D Converter such as one of Microchip's 12-bit A/D Converter, MCP320X, can be used to further enhance the circuit's accuracy.

High Precision Temperature Sensing with a 12-bit Converter

The circuit shown in Figure 15 can be further enhanced to allow for 12-bit accuracy with the addition of a MCP3201 12-bit A/D Converter and a 4th order low pass analog filter. With this circuit, the PIC12C671 is replaced with the PIC12C509.

The analog circuit in Figure 16, remains unchanged from the design shown in Figure 15 up to the analog low pass filter. This additional low pass filter is constructed using the MCP602, CMOS dual operational amplifier. The 4th order low pass filter Butterworth design that is implemented in this circuit has a cut-off frequency of 10Hz. This cut-off frequency assumes that the sample rate of the MCP3201 is 20Hz or greater. The analog filter is used to remove the instrumentation amplifier noise, as well as the noise that may be aliased into the digital conversion from the environment. This filter was designed using Microchip's FilterLab program. For more information about analog filter design, refer to Microchip's AN699, "Anti-Aliasing Filters for Data Acquisition Systems". The 12-bit resolution provided by the MCP3201 allows for a temperature measurement accuracy of 0.1 °C over the 500 °C to 1000 °C range of this circuit.

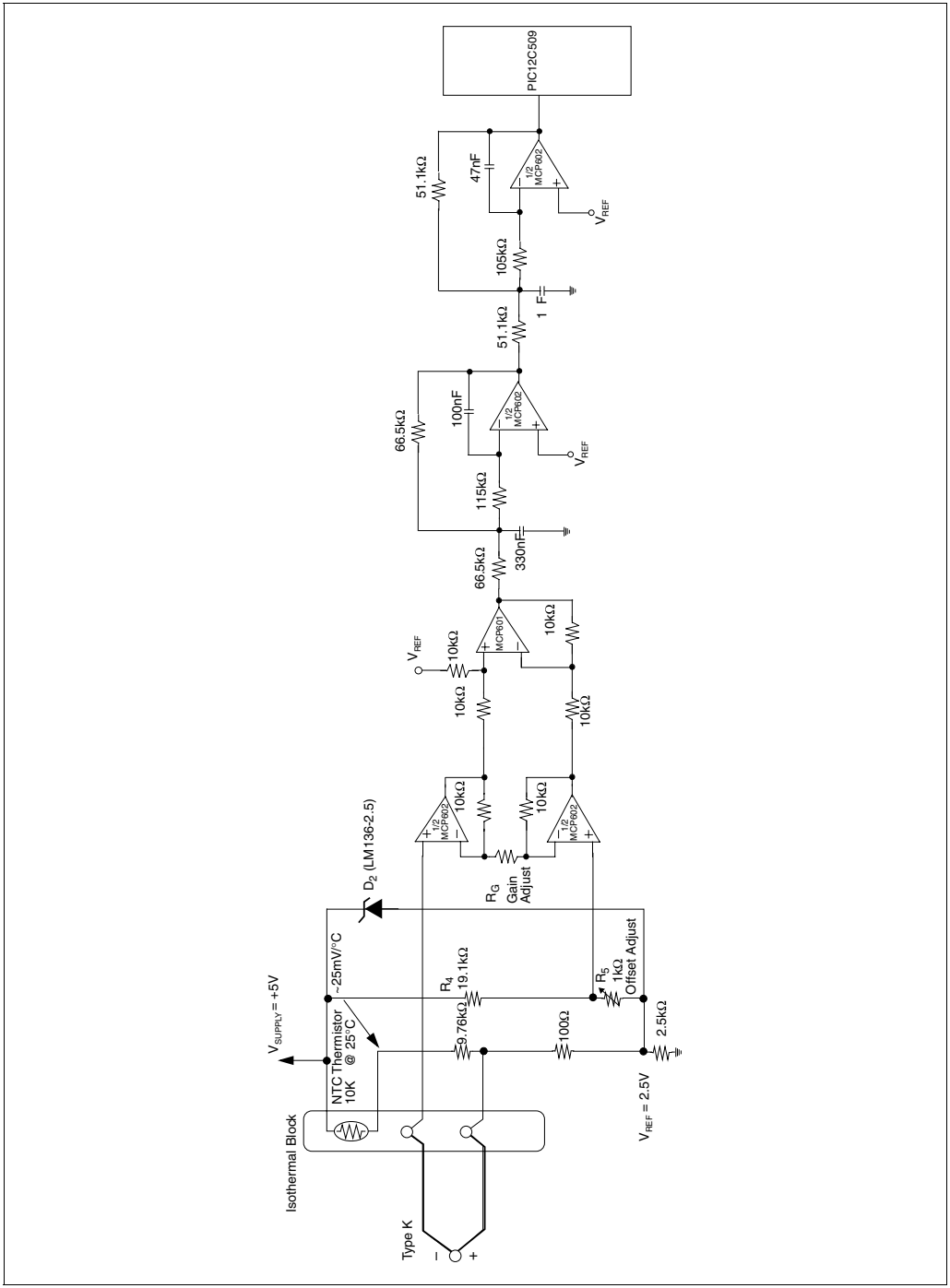


FIGURE 16: *This circuit will provide 12-bit accurate temperature sensing results using a thermocouple. In this circuit, an external A/D Converter (MCP3201), is used to digitize the analog signal.*

THERMOCOUPLE LINEARIZATION

Once a voltage from the absolute reference temperature sensor is digitized, the processor can implement a variety of algorithms. In the case with the circuit in Figure 15, the processor scans a simple look-up table. With this type of data, the microcontroller is left to translate the signal from the sensing element into the appropriate EMF voltage.

For high precision applications, look-up tables may not be adequate. In these cases, a multi-order polynomial can be used to generate the thermocouples temperature. The polynomial coefficients for Voltage to Temperature Conversion ($T = a_0 + a_1V + a_2V^2 + ... + a_nV^n$) are shown in Table 7.

For further discussion concerning the firmware implementation of thermocouple linearization, refer to AN556. This application note discussed the implementation of look-up tables. Additionally, firmware is available from Microchip that provides look-up tables code to do linearization that is directly programmable into the PICmicro® microcontroller of your choice.

CONCLUSION

Thermocouples have their advantages when used in tough application problems. They are rugged and impervious to hostile environments. The voltage output of this temperature sensing element is relatively low when compared to the devices that can convert voltage signals to a digital representation. Consequently, analog gain stages are required in the circuit.

	Thermocouple Type					
	E	J	K	R	S	T
Range	0° to 1000°C	0° to 760°C	0° to 500°C	-50° to 250°C	-50° to 250°C	0° to 400°C
a_0	0.0	0.0	0.0	0.0	0.0	0.0
a_1	1.7057035E-2	1.978425E-2	2.508355E-2	1.8891380E-1	1.84949460E-1	2.592800E-2
a_2	-2.3301759E-7	-2.00120204E-7	7.860106E-8	-9.3835290E-5	-8.00504062E-5	-7.602961E-7
a_3	6.543558E-12	1.036969E-11	-2.503131E-10	1.3068619E-7	1.02237430E-7	4.637791E-11
a_4	-7.3562749E-17	-2.549687E-16	8.315270E-14	-2.2703580E-10	-1.52248592E-10	-2.165394E-15
a_5	-1.7896001E-21	3.585153E-21	-1.228034E-17	3.5145659E-13	1.88821343E-13	6.048144E-20
a_6	8.4036165E-26	-5.344285E-26	9.804036E-22	-3.8953900E-16	-1.59085941E-16	-7.293422E-25
a_7	-1.3735879E-30	5.099890E-31	-4.413030E-26	2.8239471E-19	8.23027880E-20	
a_8	1.0629823E-35		1.057734E-30	-1.2607281E-22	-2.34181944E-23	
a_9	-3.2447087E-41		-1.052755E-35	3.1353611E-26	2.79786260E-27	
a_{10}				-3.3187769E-30		
Error	+/-0.02°C	+/-0.05°C	+/-0.05°C	+/-0.02°C	+/-0.02°C	+/-0.03°C

TABLE 7: *NIST Polynomial Coefficients of Voltage-to-temperature conversion for various thermocouple type*

REFERENCES

Baker, Bonnie, “Thermistors in Single Supply Temperature Sensing Circuits”, *AN685, Microchip Technology Inc.,* 1998

Baker, Bonnie, “Precision Temperature Sensing with RTD Circuits”, *AN687, Microchip Technology Inc.,* 1998

Baker, Bonnie, “Temperature Sensing Technologies”, *AN679, Microchip Technology Inc.,* 1998

Baker, Bonnie, “Anti-Aliasing Filters for Data Acquisition Systems”, *AN699, Microchip Technology Inc.,* 1998

Klopfenstein, Rex, “Software Linearization of a Thermocouple”, *SENSORS*, Dec. 1997, pg 40

“Practical Temperature Measurements”, *OMEGA Catalog*, pg 2-11

“Thermocouples and Accessories”, *Measurement & Control,* June 1996, pg 190

“RTD Versus Thermocouple”, *Measurement & Control,* Feb., 1997, pg 108

“Ya Can’t Calibrate a Thermocouple Junction!, *Part 2 - So What?” Measurement & Control*, Oct. 1996, pg 93

“A Comparison of Programs That Convert Thermocouple Properties to the 1990 International Temperature & Voltage Scales”, *Measurement & Control*, June, 1996, pg 104

“Thermocouple Basics”, *Measurement & Control,* June, 1996, pg 126

G.W. Burns, M.G. Scroger, G.F. Strouse, et al. Temperature-Electromotive Force Reference Functions and Tables for the Letter-Designated Thermocouple Types Based on the IPTS-90 NIST Monograph 175. Washington, D.C.: *U.S. Department of Commerce*, 1993

D’Sousa, Stan, “Implementing a Table Read”, *AN556, Microchip Technology Inc.*, 1997

NOTES:

AN685

Thermistors in Single Supply Temperature Sensing Circuits

Author: *Bonnie C. Baker*
Microchip Technology Inc.

INTRODUCTION

There is a variety of temperature sensors on the market all of which meet specific application needs. The most common sensors that are used to solve these application problems include the thermocouple, Resistive Temperature Detector (RTD) thermistor, and silicon-based sensors. For an overview and comparison of these sensors, refer to Microchip's *AN679, "Temperature Sensing Technologies"*.

This application note focuses on circuit solutions that use Negative Temperature Coefficient (NTC) thermistors in the design. The Thermistor has a non-linear resistance change-over temperature. The degree of this non-linearity will be discussed in the "Hardware Linearization Solutions" section of this application note. From this discussion, various linearization resistor networks will be shown with error analysis included. Finally, the signal conditioning path for the thermistor system will be covered with complete application circuits from sensor or microprocessor.

THERMISTOR OVERVIEW

The term "thermistor" originated from the descriptor THERMally Sensitive ResISTOR. The two basic types of thermistors are the Negative Temperature Coefficient (NTC) and Positive Temperature Coefficient (PTC). The NTC thermistor is best suited for precision temperature measurement. The PTC is best suited for switching applications. This application note will only discuss NTC applications.

The NTC thermistor is used in three different modes of operation which services a variety of applications. One of the modes exploits the resistance-versus-temperature characteristics of the thermistor. The other two modes take advantage of the voltage-versus-current and current-over-time characteristics of the thermistor.

Voltage-Versus-Current Mode

Voltage-versus-current applications use one or more thermistors that are operated in a self-heated, steady-state condition. An application example for an NTC thermistor in this state of operation would be using a flow meter. In this type of circuit, the thermistor would be in an ambient self-heated condition. The thermistor's resistance is changed by the amount of heat generated by the power dissipated by the element. Any change in the flow of the liquid or gas across the device changes the power dissipation factor of the thermistor element. In this manner, the resistance of the thermistor is changed, relative to the degree of cooling provided by the flow of liquid or gas. A useful thermistor graph for this phenomena is shown in Figure 1. The small size of the thermistor allows for this type of application to be implemented with minimal interference to the system. Applications such as vacuum manometers, anemometers, liquid level control, fluid velocity and gas detection are used with the thermistors in voltage-versus-current mode.

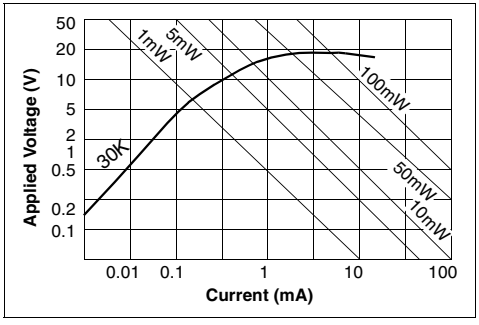


FIGURE 1: *When a thermistor is overheated by its own power, the device operates in the voltage-versus-current mode. In this mode, the thermistor is best suited to sense changes in the ambient conditions, such as changes in the velocity of air flow across the sensor.*

Current-Over-Time Mode

The current-over-time characteristics of a thermistor also depends on the dissipation constant of the thermistor package as well as element's heat capacity. As current is applied to a thermistor, the package will begin to self-heat. If the current is continuous, the resistance of the thermistor will start to lessen. The thermistor current-time characteristics can be used to slow down the affects of a high voltage spike, which could be for a short duration. In this manner, a time delay from the thermistor is used to prevent false triggering of relays.

The effect of the thermistor current-over-time delay is shown in Figure 2. This type of time response is relatively fast as compared to diodes or silicon based temperature sensors. The diode and silicon based sensors require several minutes to reach their steady state temperature. In contrast, thermocouples and RTDs are equally as fast as the thermistor, but they don't have the equivalent high level outputs. Applications based on current-over-time characteristics include time delay devices, sequential switching, surge suppression or in rush current limiting.

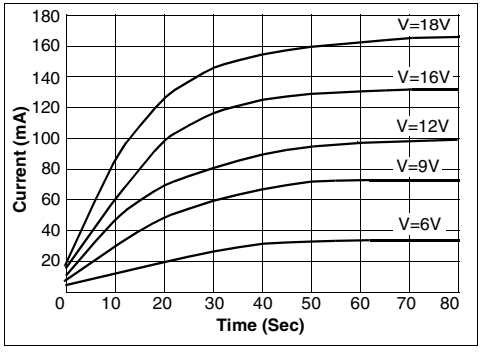


FIGURE 2: *The time constant of the thermal mass of the thermistor sensor can be used to time delay a reaction to changes in conditions in a circuit. If a thermistor is overdriven, the thermal mass time constant of the sensor eventually causes the thermistor to overheat, reducing its resistance.*

Resistance-Versus-Temperature Mode

By far, applications using the first mode, resistance-versus-temperature, NTC Thermistor configurations, are the most prevalent. These circuits perform precision temperature measurement, control and compensation. Unlike applications that are based on the voltage-versus-current and current-over-time characteristics of the thermistor, the resistance-versus-temperature circuits depend on the thermistor being operated in a "zero-power" condition. This condition implies that there is no self-heating of the thermistor as a consequence of current or voltage excitation. The resistance-versus-temperature response of a 10kΩ, NTC thermistor is shown in Figure 3.

The resistance across the thermistor is relatively high in comparison to the RTD element which is usually in the hundreds of ohms range. Typically, the 25°C rating for thermistors is from 1kΩ up to 10MΩ. The housing of the thermistor varies as the requirements for hermeticity and ruggedness vary, but in all cases, there are only two wires going to the element. This is possible because of the resistance of the wiring over temperature is considerably lower than the thermistor element. Consequently, a four wire configuration is not necessary, as it is with the RTD element. (Refer to *AN687, "RTD Temperature Sensing Circuits"* for details.)

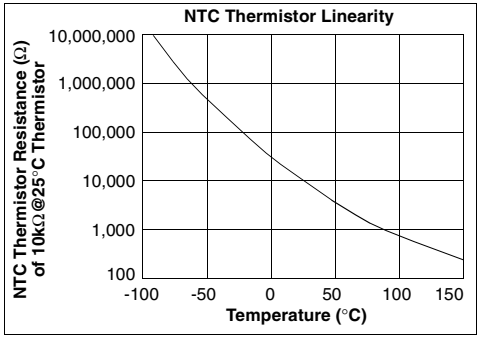


FIGURE 3: *In precision temperature measurement environments, the thermistor is used in a "zero power" condition. In this condition, the power consumption of the thermistor has a negligible affect on the elements resistance. This is a graph of an NTC 10k*Ω *thermistor resistance-versus-temperature.*

Since the thermistor is a resistive element, current excitation is required. The current can originate from a voltage or current reference, as will be shown in the "Hardware Linearization Solutions" section of this application note. The performance of the thermistor in Figure 3 is fairly repeatable as long as the power across the device does not exceed the power dissipation capability of the package. Once this condition is violated, the thermistor will self-heat and artificially decrease in resistance, giving a higher than actual temperature reading.

Figure 3 illustrates the high degree of non-linearity of the thermistor element. Although the thermistor has considerably better linearity than the thermocouple linearity, the thermistor still requires linearization in most temperature sensing circuits. The non-linear response of the thermistor can be corrected in software with an empirical third-order polynomial or a look-up table. There are also easy hardware lineariza tion techniques that can be applied prior to digitalization of the output of the thermistor. These techniques will be discussed later in this application note. The third-order polynomial is also called the Steinhart-Hart Thermistor equation. This equation is an approximation and can replace the exponential expression for a thermistor. Wide industry acceptance makes it the most useful equation for precise thermistor computation.

The Steinhart-Hart equation is:

$$T = 1/(A_0 + A_1(\ln R_T) + A_3(\ln R_T^3))$$

$$\ln R_T = B_0 + B_1/T + B_3/T_3$$

where:

T is the temperature of the thermistor in Kelvin.

A_0, A_1, A_3, B_0, B_1, and B_3, are contents provided by the thermistor manufacturer.

R_T is the thermocouple resistance at temperature, T.

With a typical thermistor, this third-order linearization formula provides ±0.1°C accuracy over the full temperature range. This is usually better than the accuracy of individual elements from part to part.

Although the temperature range of the thermistor is a little better than the diode or silicon-based temperature sensor (–55°C to +175°C), it is still limited to a practical range of –100°C to +175°C. This can also be compared to the temperature sensing range of the RTD (–200°C to 600°C) or the thermocouple which ranges up to 1820°C.

The advantages versus disadvantages of the thermistor are summarized in Table 1.

ADVANTAGES	DISADVANTAGES
Fast	Non-Linear
Small	Excitation Required
Two-Wire	Limited Temperature Range
Inexpensive	Self-Heating
	Fragile

TABLE 1: *Summary of Thermistor Advantages and Disadvantages.*

Thermistors are manufactured by a large variety of vendors. Each vendor carefully specifies their thermistor characteristics with temperature, depending on their manufacturing process. Of all of the temperature sensors, the thermistor is the least expensive sensing element on the market. Prices start at $0.10 with some vendors and range up to $25.

The thermistor is used in a large variety of applications such as automotive monitor and control exhaust emissions, ice detection, skin sensors, blood and urine analyzers, refrigerators, freezers, mobile phones, base stations laser drives, and battery pack charging. In the precision instrumentation applications, thermistors are used in hand-held meters and temperature gauges.

THE TEMPERATURE- RESISTIVE MODE OF THE THERMISTOR

An electrical configuration for the thermistor is shown in Figure 4. This illustrates a seemingly obvious way to excite the thermistor and measure the change in resistance where the sensing element is excited with a current source.

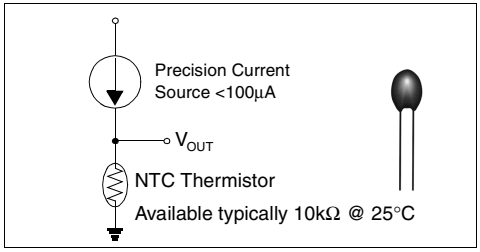


FIGURE 4: *Common sense would dictate that the thermistor be excited by a precision constant current source as shown in this figure. A picture of an NTC Thermistor is shown on the right.*

With this style of excitation, the magnitude of the current source is typically below 100µA, preferably 20µA. Lower currents prevent the thermistor from entering a self-heating condition as described previously. This style of excitation is effective for sensing a limited range of temperatures. Larger ranges of temperature have deltas in resistance that are too high to accurately convert the resistance to voltage without bumping into the noise limitations of the analog signal path.

As an example, the temperature range of a typical thermistor from BetaTHERM is –80°C to 150°C. The change is resistance for a 10kΩ @ 25°C thermistor from BetaTHERM over its temperature range is shown in Table 2.

It is useful to note that the differential resistance for a 10°C delta at high temperature is significantly smaller than a 10°C delta at low temperatures. For instance, the change in resistance of the device in Table 2 from 125°C to 135°C is 76.28Ω (340.82Ω – 264.54Ω). The change in resistance of the same thermistor from –25°C to –15°C is 58.148kΩ. This diversity in the ratio of resistance to temperature over the range of thermistor creates an awkward analog problem. If the thermistor in this example is excited with a 20µA current source, the analog circuit must discriminate between 0.015V deltas at high temperatures and 1.16V deltas at low temperatures for Δ10°C of resolution. This forces the LSB size in a linear digitizing system to be 1/2 of 0.015V. This would require a 9.57-bit system to achieve 10°C accuracy from the system over a temperature span of -25°C to 135°C (delta of 160°C).

Temp (°C)	R Value (Ω)	Temp (°C)	R Value (Ω)	Temp (°C)	R Value (Ω)
-80	7296874	0	32650.8	75	1480.12
-75	4713762	5	253985.5	80	1256.17
-70	3095611	10	19903.5	85	1070.58
-65	2064919	15	15714.0	90	916.11
-60	1397935	20	12493.7	95	786.99
-55	959789	25	10000	100	678.63
-50	667828	30	8056.0	105	587.31
-45	470609	35	6530.1	110	510.06
-40	335671	40	5324.9	115	44.48
-35	242195	45	4366.9	120	388.59
-30	176683	50	3601.0	125	340.82
-25	130243	55	2985.1	130	299.82
-20	96974	60	2487.1	135	264.54
-15	72895	65	2082.3	140	234.08
-10	55298	70	1751.6	145	207.70
-5	42314.6			150	184.79

TABLE 2: *Resistive changes with temperature of a BetaTHERM, 10kΩ @ 25 °C (10K3A1) NTC Thermistor in its "zero power" mode.*

LINEARIZATION SOLUTIONS

It is obvious in this example that the conversion process is inefficient if a linear response is required. It is also obvious that the digital output word will require a look-up table to linearize the response. Additionally, temperature accuracy is usually required for most systems. These problems can be solved to a small degree by using a high resolution Analog-To-Digital (A/D) Converting device. In this scenario, bits will still be thrown away, but the LSB size is smaller. An alternative is to implement linearization with the analog hardware.

A simple approach to a first level linearization of the thermistor output is to use one of the three circuits shown in Figure 5. In Figure 5a. the thermistor is placed in series with a standard resistor (1%, metal film) and a voltage source. The temperature response and linearity of the system shown in Figure 5a. is shown in Figure 6. In this figure, the series thermistor system responds to temperature in a linear manner over a limited temperature range. The linearization resistor's value (R_{SER}) should be equal to magnitude of the thermistor at the mid-point of the temperature range of interest. This creates a response where the output slope of the resistive network is at its steepest at this mid-point temperature. If high precision is required, this range is typically +/-25°C around the nominal temperature of the thermistor at the R_{SER} value.

In Figure 5b., the thermistor is placed in parallel with a standard resistor (R_{PAR}), which creates a composite resistor element. This type of resistive configuration is typically used in system feedback loops and used for automatic gain control circuits.

The resistance to temperature response along with the linearization error of this circuit configuration is shown in Figure 7. Once again, the optimum linearity response of this resistive network is obtained at the point where the thermistor resistance and R_{PAR} are equal.

A third linearization approach is shown in Figure 5c. This circuit combines the parallel configuration in Figure 5b. with an additional reference resistor and a capacitor. The switchable reference is used to charge and discharge the parallel NTC resistance and the reference resistor against the integrating capacitor, C_{INT}. With this circuit, the NTC resistance is biased to a voltage reference and the integrating capacitor charges.

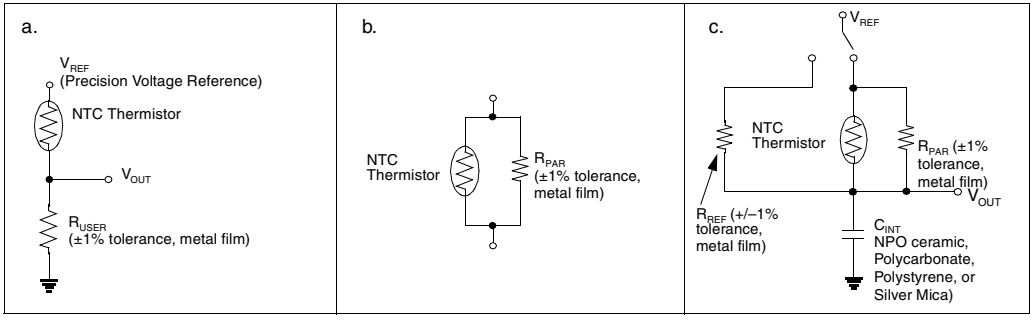


FIGURE 5: *The series configuration (a) requires a voltage excitation. The parallel configuration (b) can be used in the feedback loop of an amplifier and does not require a precision source. The parallel configuration can be combined with a capacitor (c) which provides a linear circuit response with time.*

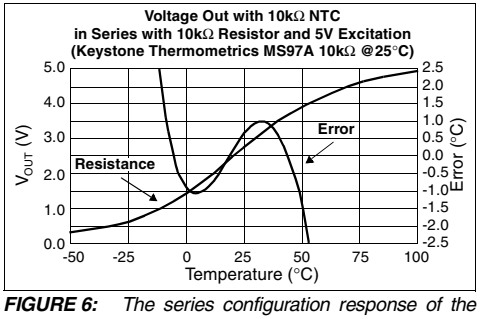


FIGURE 6: *The series configuration response of the circuit shown in Figure 5a. has good linear response in a ±25°C range surrounding the temperature where both resistors (NTC and R_{SER}) are equal. The error in this range is typically within ±1%. VREF = 5V.*

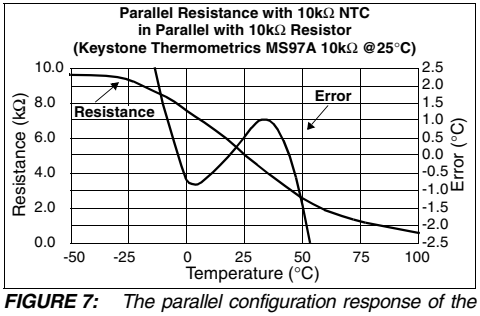


FIGURE 7: *The parallel configuration response of the circuit shown in Figure 5c. allows for a counter to be used to determine the relative resistance of the NTC element.*

Once the voltage at the top of the integrating capacitor reaches a threshold value V_{TH} (Figure 8), the integration time is recorded and the switching voltage reference is set to zero which discharges the integrating capacitor.

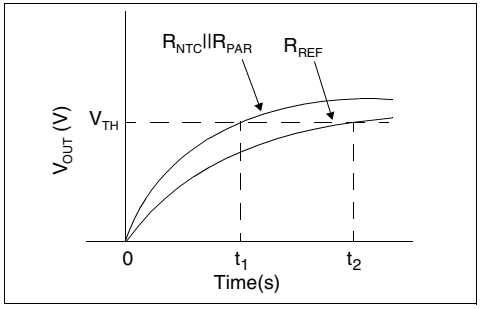


FIGURE 8: *The RC time response of the circuit shown in Figure 5c. allows for the microcontroller counter to be used to determine the relative resistance of the NTC element.*

Once the integrating capacitor is discharged, the reference voltage is applied to the reference resistor R_{REF}. This circuit is allowed to integrate until V_{OUT} reaches V_{TH} and the time of that integration period is recorded.

The integration time of this circuit can be calculated using:

$$V_{OUT} = V_{REF}\,(1 - e^{-t/RC})\ or$$
$$t = RC\ ln\ (1 - V_{TH}/V_{REF})$$

If the ratio of V_{TH}:V_{REF} is kept constant, the unknown resistance of the R_{NTC} || R_{PAR} can be determined with:

$$R_{NTC}||R_{PAR} = (t_2/t_1) \times R_{REF}$$

In this configuration, the resistance calculation of the parallel combination of R_{NTC} || R_{PAR} is independent of C_{INT}.

The implementation of this linearization circuit will be discussed with further detail in the “Thermistor Signal Conditioning Circuits” of this application note.

The circuits in Figure 5, along with the other configurations shown in Figure 9 linearize the thermistor to various ways. Figure 9a. uses the combination of the parallel and serial configurations shown in Figure 5 to extend the linear temperature response beyond 50°C. Figure 9b. demonstrates a way that the initial DC voltage of a thermistor linearization circuit can be removed by employing a bridge configuration. The circuit in Figure 9c. uses a switching network to adjust the linearization range of the of the NTC Thermistor. Additionally, there is a resistor divider added that implements a bridge configuration in order to reduce DC errors. The response of all of these networks can easily be modeled in an excel spreadsheet or mathcad which can be used to generate the appropriate look-up tables.

The next section of this application note will use the networks in Figure 5 to implement complete application circuits.

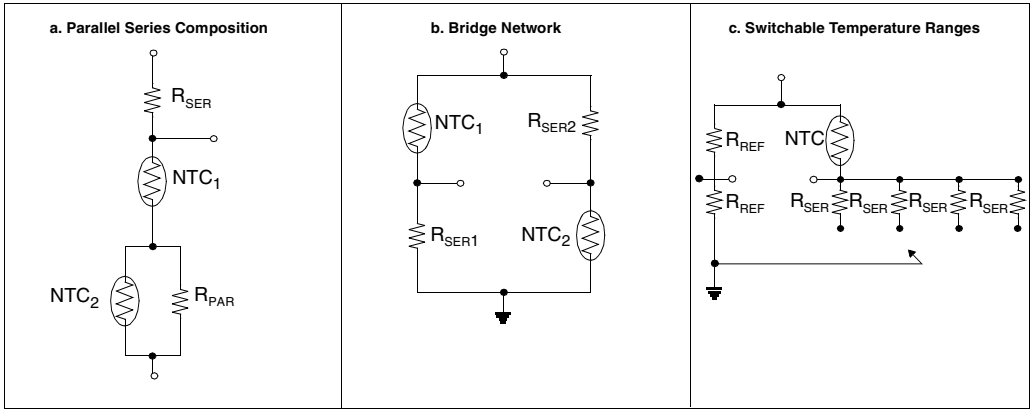


FIGURE 9: *Other Thermistor Linearization Circuits.*

THERMISTOR SIGNAL CONDITIONING CIRCUITS

There is a large variety of application circuits where the thermistor can be utilized. The three circuits in this application note use the thermistor to implement the cold junction compensation portion of a thermocouple circuit, a linear variable gain versus temperature circuit and an integrated scheme which achieves high accuracy.

Thermocouple Cold Junction Compensation

Although thermocouples can sense temperatures accurately at extreme temperatures or in ambient hostile conditions, a reference temperature is required, if an absolute temperature measurement is desired. (See Microchip's *AN684, "Single Supply Temperature Sensing with Thermocouples"* for details concerning thermocouple circuit requirements.)

The circuit in Figure 10 is designed to sense the temperature at the isothermal block location with a thermistor. The linearized temperature response of the thermistor is divided down to appropriate levels in order to minimize the EMF voltage errors introduced to the circuit by the parasitic thermocouples on the isothermal block. This style of compensation is done in hardware, requiring no supportive firmware compensation schemes.

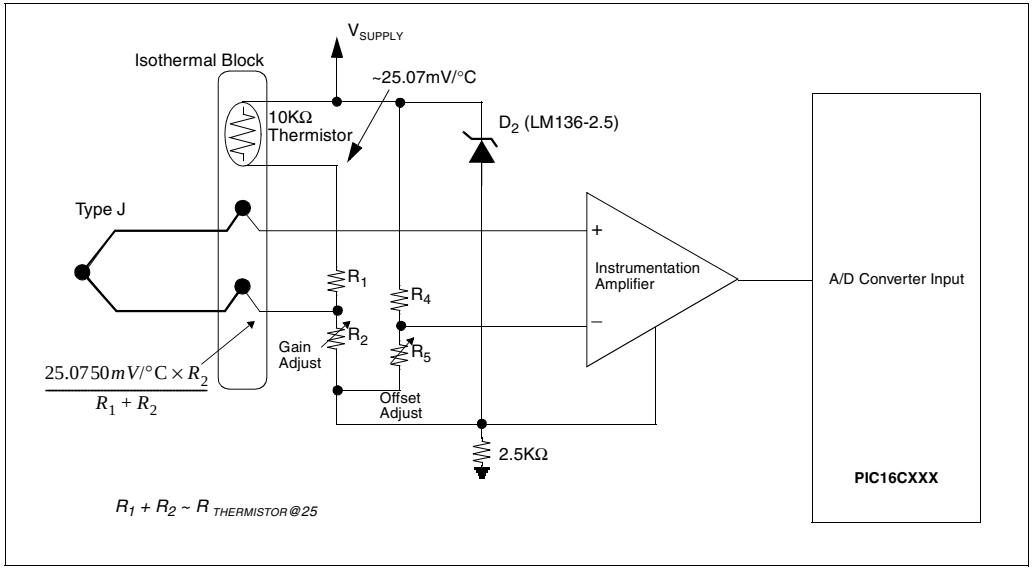


FIGURE 10: *A thermistor is used to sense the temperature of the isothermal block in a thermocouple temperature sensing application.*

The drift with temperature of the parasitic thermocouples on the isothermal block is approximately –51µV/°C. The thermistor circuit changes by 25.07mV/°C over the 0 to 50°C linear range given the resistor configuration and the 2.5V excitation voltage. The thermistor drift is divided down using the resistor divider formed with R_1 and R_2. Appropriate resistor values for R_1 and R_2 with a Type J thermocouple is 100Ω and 49.9kΩ, inclusive. The R_4 and R_5 resistor divider is used to zero offsets in the system as well as implement any required level shifts.

An instrumentation amplifier is used to differentiate the offset error correction circuitry and the Type J thermocouple EMF voltage. (For more details about instrumentation amplifiers, see Microchip's *AN682, "Using Operational Amplifiers for Analog Gain in Embedded System Design".*)

With the thermistor linearization circuitry in place, the voltage changes at the input to the instrumentation amplifier in accordance with temperature changes at the Type J thermocouple measurement site.

The instrumentation amplifier is configured in the appropriate gain for the expected temperature excursions of the Type J thermocouple. The output of the gained analog signal is digitized and used by the microcontroller. With this circuit implementation, the microcontroller is only required to linearize the thermocouple output response.

Temperature Dependent Reference

A temperature dependent reference voltage can be constructed using thermistor/resistive parallel combination illustrated in Figure 5b. as the feedback element in an operational amplifier circuit. The implementation of this type of circuit configuration is shown in Figure 11. In this circuit, a precision reference is used to drive the inverting input of an operational amplifier. The gain of the amplifier portion of the circuit is:

$$V_{OUT:AMP} = V_{IN:AMP}(1 + (R_{NTC}||R_{PAR})/R_1)$$

where:

$V_{OUT:AMP}$ is the voltage at the output of the operational amplifier

$V_{IN:AMP}$ is the voltage presented to the non-inverting input of the amplifier

A 2.5V precision voltage reference is used to generate the 0.276 voltage at the input to the operational amplifier. When the temperature of the NTC thermistor is equal to 0°C, the resistance of the thermistor is approximately 32,650.8Ω. The value of the parallel combination of this resistor and the 10kΩ metal film resistor (R_{PAR}) is equal to 7655.38Ω. This gives a operational amplifier gain of 14.94 V/V or an output voltage ($V_{OUT:AMP}$) of 4.093V.

When the temperature of the NTC thermistor is 50°C, the resistance of the thermistor is approximately 3601Ω. Following the same calculations above, the operational amplifier gain becomes 5.8226V/V, giving a 1.595V at the output of the amplifier.

The voltage at the output of the amplifier is used as the voltage reference of a 12-bit A/D Converter. Over the reference range of 4.093V to 1.595V the converter provides 11.75-bit accurate conversions. The converter digitizes the input signal in accordance with the transfer function:

$$DIGITAL\ OUT = \left(\frac{V_{IN:ADS}}{V_{OUT:AMP}}\right)(2^{12} - 1)$$

(to the nearest integer value)

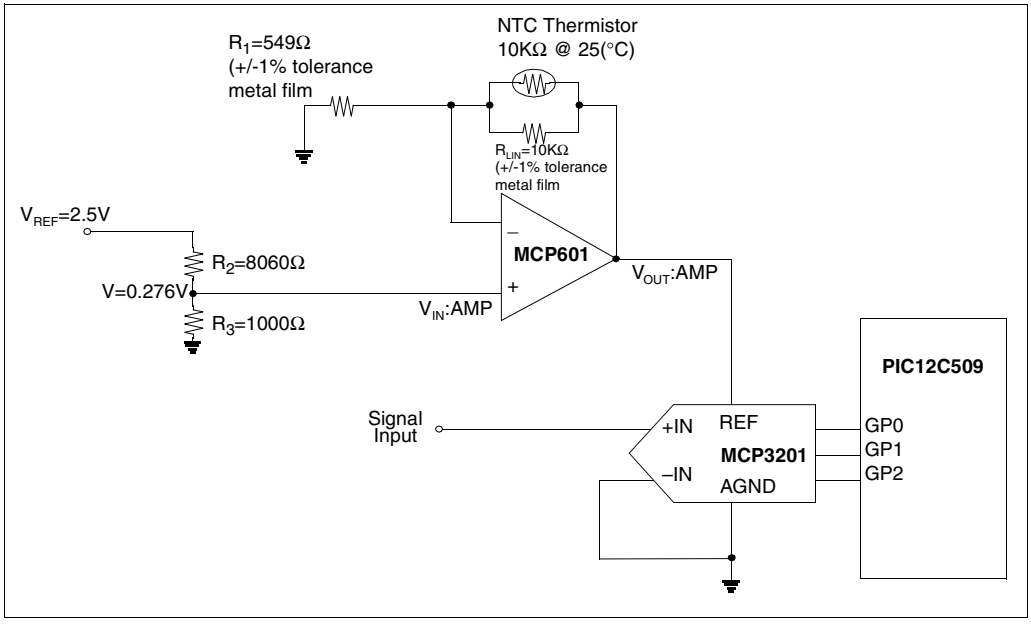


FIGURE 11: *A thermistor is used to change the gain of an amplifier circuit with respect to temperature.*

Temperature Sensing Using an Integrator

The linearization circuit in Figure 5c. is simply implemented with one microcontroller in the signal path as shown in Figure 12.

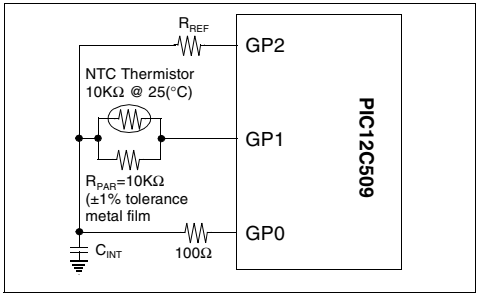


FIGURE 12: *This circuit switches the voltage reference on and off at GP1 and GP2. In this manner, the time constant of the NTC Thermistor (R_{NTC} || R_{PAR}) and integrating capacitor (C_{INT}) is compared to the time constant of the reference resistor (R_{REF}) and integrating capacitor.*

This sensing circuit is implement by setting GP1 and GP2 of the PIC12C509 as inputs. Additionally, GP0 is set low to discharge the capacitor, C_{INT}. Once C_{INT} is discharged, the configuration of GP0 is changed to an input and GP1 is set to a high output. A timer counts the amount of time before GP0 changes to 1, giving the time, t_1 per Figure 8.

At this point, GP1 and GP2 are again set as inputs and GP0 as an output low. Once the integrating capacitor C_{INT}, has time to discharge, GP2 is set to a high output and GP0 as an input. A timer counts the amount of time before GP0 changes to 1, giving the time, t_2, per Figure 8.

For more details concerning the implementation of this type of integrating circuit, refer to Microchip's *AN512, "Implementing Ohmmeter/Temperature Sensor"*, and *AN611, "Resistance and Capacitance Meter Using a PIC16C622"*.

CONCLUSION

Although the thermistor is non-linear, it can be tamed for a limited temperature range. This allows the design of an inexpensive temperature sensing device which can be used in a variety of Analog-to-Digital Converter applications.

REFERENCES

Lavenuta, Greg, "Negative Temperature Coefficient Thermistors – the Temp Calibration Standard", *SENSORS*, *August, 1997, pg 54.*

Lavenuta, Greg, "Negative Temperature Coefficient Thermistors – Level of Uncertainty", *SENSORS*, *June 1997, pg 47.*

Lavenuta, Greg, "Negative Temperature Coefficient Thermistors – Measuring", *SENSORS*, *Sept, 1997, pg 48.*

Lavenuta, Greg, "Negative Temperature Coefficient Thermistors" *SENSORS*, *May 1997, pg 46.*

Lavenuta, Greg, "Negative Temperature Coefficient Thermistors – Temp Controlled Bath", *SENSORS*, *July 1997, pg17.*

Paillard, Bruno, "Temperature Compensating an Integrated Pressure Sensor", *SENSORS*, *Jan. 1998, pg 36.*

Thermometrics Corporation, *Catalog, 1996.*

Baker, Bonnie, "Temperature Sensing Technologies", *AN679*, *Microchip Technology Inc., 1998.*

"Practical Temperature Measurements", O*MEGA CATALOG*, *pg Z-11.*

Baker, Bonnie, "RTD Temperature Sensing Circuits", *AN687*, *Microchip Technology Inc., 1998.*

Baker, Bonnie, "Single Supply Temperature Sensing with Thermocouples", *AN684*, *Microchip Technology Inc., 1998.*

Baker, Bonnie, "Using Operational Amplifiers for Analog Gain in Embedded System Design", *AN682, Microchip Technology Inc., 1998.*

Cox, Doug, "Implementing Ohmmeter/Temperature Sensor", *AN512*, *Microchip Technology Inc.*

Richey, Rodger, "Resistance and Capacitance Meter Using a PIC16C622", *AN611*, *Microchip Technology Inc.*

NOTES:

AN686

Understanding and Using Supervisory Circuits

Author: *Bruce Negley*
Microchip Technology Inc.

SCOPE

This application note discusses what microcontroller supervisory devices are, why they are needed and some factors to consider when choosing one. Supervisory devices is a broad term that encompasses POR (power on reset) devices, BOD (brown-out detect) devices and watchdog timer devices. This application note will cover supervisor devices with POR and BOD functions only.

WHAT DOES A SUPERVISORY CIRCUIT DO?

A supervisory circuit can be used for several different applications, but there are two primary functions that a supervisor provides:

1. During a power up sequence, the device holds a microcontroller in reset until the system power has come up to the correct level and stabilized (the POR function), and
2. reset the controller immediately if the power drops below a nominal value either at power down or during a 'brown-out' condition.

Some supervisor devices also provide things like low battery warning, watchdog timer and other more elaborate functions that are beyond the scope of this application note.

WHY DO I NEED A SUPERVISORY CIRCUIT ANYWAY?

One question system designers may ask themselves is, "Why do I need one of these things anyway?" There are 3 situations that you must consider when answering this question:

1. What would happen to the microcontroller (or other devices in the system) if there was noise on the supply voltage as it powers up?
2. What would happen if there is a glitch on the power supply while the system is running?
3. What does the microcontroller do when the system power is turned off?

If you ponder these questions and have visions of phone calls from angry customers, then you might consider using a supervisor device.

IN THE BEGINNING: POWER-UP PROBLEMS

Most designers working on a prototype system are familiar with putting a reset switch of some kind on the reset pin of the microcontroller. Why? Because they are making both hardware and firmware changes, which sometimes cause the system to malfunction, resulting in the microcontroller no longer behaving in a rational manner. Sometimes it just plain doesn't work. The system designer pushes the reset button a couple of times to determine if the problem goes away. If not, more changes are made and the process continues. The push button provides a means of manually resetting the system. This may work fine for the system development phase, but what do you do to ensure proper system power-up when it goes into production?

Many systems rely on a simple pullup resistor tied to the reset line and their system works fine every time. But what if different components in the system are all powering up as the supply voltage is ramping up and noise is injected onto the supply line? Most microcontrollers have specs that describe power up ramps for proper initialization of the controller. A glitch on the supply line may very well cause the microcontroller (or some other component) to power-up incorrectly and prevent the system from operating as intended. See Figure 1. A supervisor device solves this problem by holding the microcontroller in reset until the power has reached a stable level. Timeout periods vary for different devices but typical values are 150ms - 500ms. When the timeout period is complete, the device will release the reset line and allow the microcontroller to begin exection of its code.

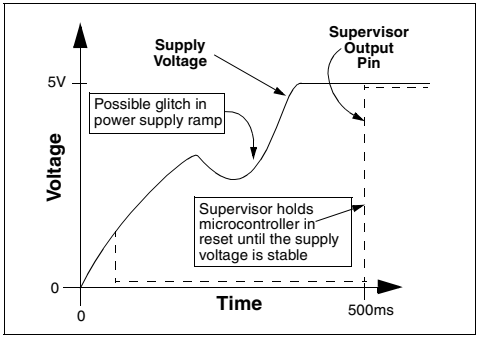


FIGURE 1: *POR Function*

Brown-Out: A Dirty Little Problem

Brown-out (Figure 2) is a condition where the supply voltage dips or 'sags' down to a safe operating level before returning to a nominal level. This condition can be caused by many different things such as inadequate power regulation, system components turning on or off, system malfunctions, etc. Unfortunately, brown-out conditions often don't show up in the system development stage, but wait until the production run begins with all the system components installed to show their ugly heads. It is often at this point that perplexing problems are discovered, and eventually tracked down to some kind of brown-out condition. These problems can manifest themselves in many different ways including logic levels being misinterpreted or high current situations by creating invalid CMOS input levels. It is also possible to cause a more insidious problem of corrupting RAM locations inside the microcontroller. This problem can lead to irrational behavior on the part of the microcontroller that does different things at different times and may not show itself at all when an emulator is used to track down the problem.

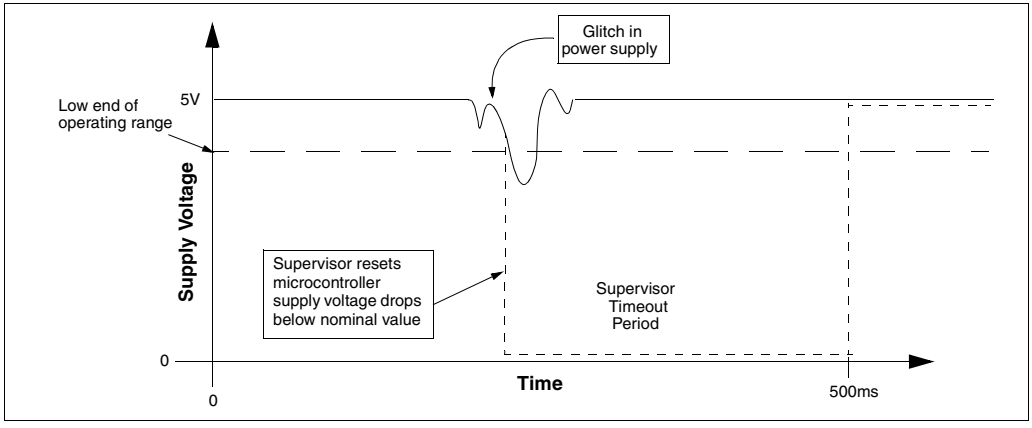


FIGURE 2: *Brown-Out Condition*

Problems at Power-Down

Most microcontrollers today do not have any on-board POR/BOD protection. Some of them do, but they may not offer adequate protection against some system failures. One system problem that is seen quite frequently is the "Microcontroller running amok" problem that occurs when the supply voltage is ramped down very slowly, such as when a bench power supply is turned down manually or during the decay of a battery supply. When this situation occurs, it is possible for many microcontrollers to begin running through its code in a somewhat random manner. There may not be enough voltage to sustain RAM locations, so the program counter as well as any other variable stored in RAM may not contain valid data. This provides the means for the micro to execute any or all portions of the code stored in program memory with indeterminate values in all RAM locations.

Obviously, the longer it takes for the supply to ramp down the greater the danger of this situation occurring and causing problems. See Figure 3. For some systems, this situation may not cause any problems more serious than some spurious data sent to a display as the system is powered down. However, if the system contains other components that work to a lower voltage such as EEPROM devices, the problem becomes potentially more serious. EEPROM devices are available on the market that work down to 1.8V and may respond to commands as low as 1.2V. If the microcontroller executes a portion of its code that controls writing to the EEPROM, then there is the distinct possibility that random data will be written to the EEPROM device, which may or may not be discovered when the system is powered up the next time. This problem very often does not show up in the system development phase because the system is not being powered up and down on a regular basis, or it is powered from a supply different from the one used in production. It often shows up when the system goes into production and the system is being tested at different stages of the production line with different power supplies. A typical scenario: Data is written into the EEPROM and the system is tested as good and then powered down. At the next station it is discovered that the EEPROM data has been corrupted. This often results in a call to the EEPROM vendor with complaints of data retention problems, when the actual problem was the microcontroller sending write commands to the EEPROM during power down.

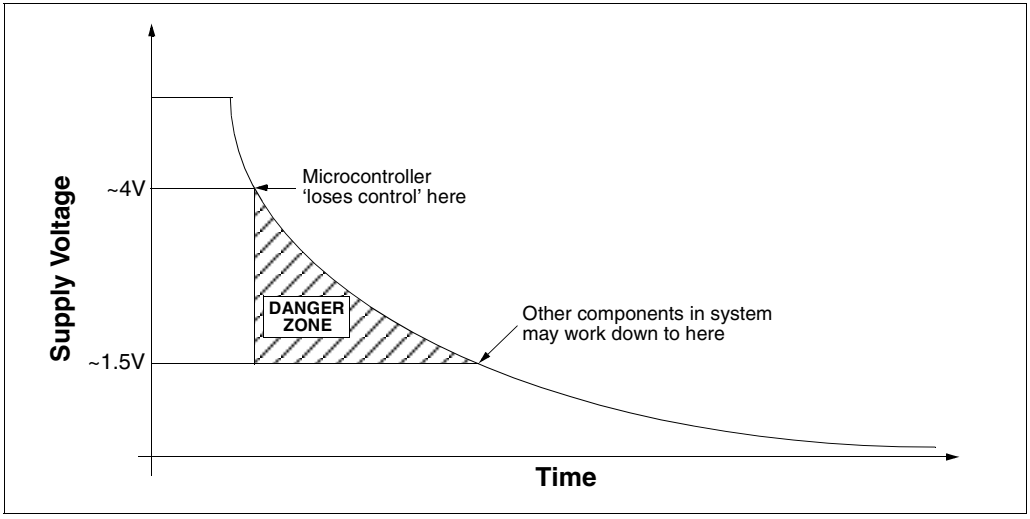


FIGURE 3: *Microcontroller loses control with slowly decaying supply*

SO HOW DO I CHOOSE THE RIGHT DEVICE?

For the standard POR/BOD type of supervisor device, there are really only a couple of factors that you need to consider when making your choice. The major factors to consider are: reset voltage, output driver type, and reset polarity. Most supervisor devices come in a variety of reset voltages to support both 5V and 3V systems. Table 1, below shows typical reset voltage ranges. Choosing the correct trip point depends mainly on the operating range of the controller you are using and the variation of your supply voltage. You want to choose the highest trip point you can that will not interfere with the normal variations of your supply voltage. For a typical microcontroller, it might operate at 5V ±10% or 4.5V - 5.5V. Choosing a device with a trip point range of 4.5V - 4.75V will ensure that the controller is reset before the low end of the operating range is reached.

Minimum Trip Point (V)	Typical Trip Point (V)	Maximum Trip Point (V)
2.55	2.625	2.7
2.85	2.925	3.0
3.0	3.075	3.15
4.25	4.375	4.50
4.35	4.475	4.60
4.50	4.625	4.75
4.60	4.725	4.85

TABLE 1: *Typical Trip Point Values*

Many vendors also provide different output driver options for their devices. The usual choices are open drain, open drain with internal pull-up and standard push-pull output drivers. The open drain options allow more than one source to pull the reset line to the reset state, such as a pushbutton or some other component that has the ability to reset the controller such as an over-temperature safety switch.

Since some microcontrollers have a low active reset line and some are high active, you must also choose a reset device with the correct polarity. For reference, the MCP100/120/130 are all active low devices and the MCP101 is active high.

CONCLUSIONS

Using supervisory circuits can protect microcontroller based systems from a number of power-related problems. If you are experiencing problems in your system that are not making sense, it may be power related and if so, it may be beneficial to add a supervisory device to the system. This application note provides some guidelines that you can use in determining what the problem might be and what device should be chosen to solve the problem.

NOTES:

AN687

Precision Temperature Sensing with RTD Circuits

Author: *Bonnie C. Baker*
Microchip Technology Inc.

INTRODUCTION

One of the most widely measured phenomena in the process control environment is temperature. Common elements such as Resistance Temperature Detectors (RTDs), thermistors, thermocouples or diodes are used to sense absolute temperatures as well as changes in temperature. For an overview and comparison of these sensors, refer to Microchip's AN679, *"Temperature Sensing Technologies*".

Of these technologies, the platinum RTD temperature sensing element is the most accurate and stable over time and temperature. RTD element technologies are constantly improving, further enhancing the quality of the temperature measurement (see Figure 1). Typically, a data acquisition system conditions the analog signal from the RTD sensor, making the analog translation of the temperature usable in the digital domain.

This application note focuses on circuit solutions that use Platinum RTDs in the design. Initially, the RTD temperature sensing element will be compared to the negative temperature coefficient (NTC) thermistor, which is also a resistive temperature sensing element. In this forum the linearity of the RTD will be presented along with calibration formulas that can be used to improve the off the shelf linearity of the element. If more information is needed concerning the thermistor temperature sensor, refer to Microchip's AN685, *"Thermistors in Single Supply Temperature Sensing Circuits"*. Finally, the signal conditioning path for the RTD system will be covered with complete application circuits from sensor or microprocessor.

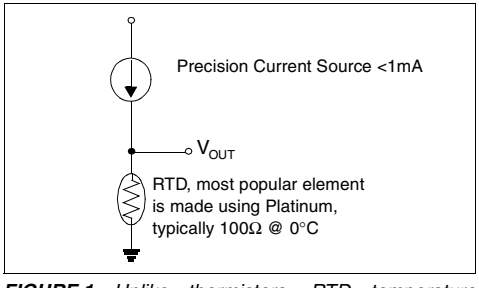


FIGURE 1: *Unlike thermistors, RTD temperature sensing elements require current excitation.*

RTD OVERVIEW

The acronym "RTD" is derived from the term "Resistance Temperature Detector". The most stable, linear and repeatable RTD is made of platinum metal. The temperature coefficient of the RTD element is positive. This is in contrast to the NTC thermistor that has a negative temperature coefficient as shown graphically in Figure 2. An approximation of the platinum RTD resistance change over temperature can be calculated by using the constant 0.00385Ω/Ω/°C. This constant is easily used to calculate the absolute resistance of the RTD at temperature.

$$RTD(T) = RTD_0 + T \times RTD_0 \times 0.00385\Omega/\Omega/^\circ C$$

where

$RTD(T)$ is the resistance value of the RTD element at temperature (Celsius),

RTD_0 is the specified resistance of the RTD element at 0°C, and

T is the temperature environment that the RTD is placed (Celsius).

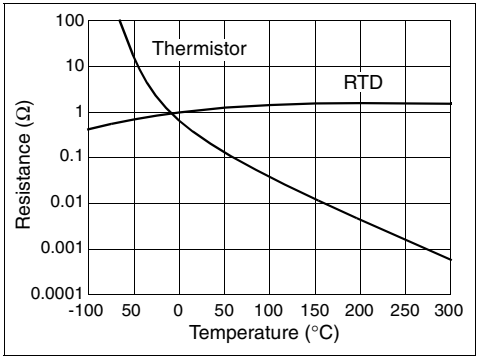


FIGURE 2: *The temperature versus resistance characteristics of the RTD sensing element is considerably different than the thermistor sensor element. The RTD sensing element has a positive temperature coefficient and is considerably more linear.*

The RTD element resistance is extremely low as compared to the resistance of an NTC thermistor element which ranges up to 1MΩ at 25°C. Typical specified 0°C values for RTDs are 50, 100, 200, 500, 1000 or 2000Ω. Of these options, the 100Ω platinum RTD is the most stable over time and linear over temperature.

If the RTD element is excited with a current reference at a level that does not create a error due to self-heating, the accuracy can be ±4.3°C over its entire temperature range of -200°C to 800°C. If a higher accuracy temperature measurement is required, the linearity formula below (Calendar-Van Dusen Equation) can be used in a calculation in the processor engine or be used to generate a look-up table.

$$RTD(T) = RTD_0(1 + AT + BT^2 - (100CT^3 + CT^4))$$

where

$RTD(T)$ is the resistance of the RTD element at temperature,

RTD_0 is the specified resistance of the RTD element at $0°C$,

T is the temperature that is applied to the RTD element (Celsius), and

A, B, and C are constants derived from resistance measurements at 0°C, 100°C, and 260°C.

The linearity performance of a typical RTD is shown in Figure 3.

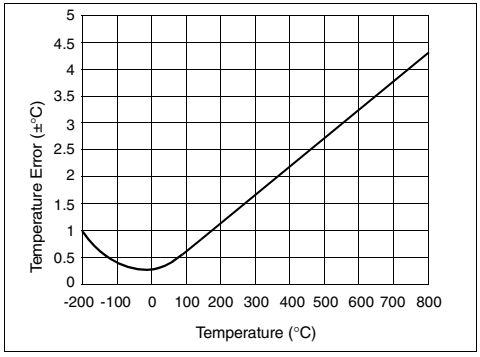


FIGURE 3: *The linearity error of the Platinum RTD temperature sensor is small when compared to other sensors such as the thermocouple and thermistor.*

The RTD element requires a current excitation. If the magnitude of the current source is too high the element will dissipate power and start to self-heat. Consequently, care should be taken to insure that ð1mA of current is used to excite the RTD element.

The advantages and disadvantages of the RTD temperature sensing element is summarized in Table 1.

ADVANTAGES	DISADVANTAGES
Very Accurate and Stable	Expensive Solution
Fairly Linear to ±4%°C	Requires Current Excitation
Good Repeatability	Self-Heating
	Low Resistive Element

TABLE 1: *RTD temperature sensing element advantages and disadvantages.*

RTD CURRENT EXCITATION CIRCUIT

For best linearity, the RTD sensing element requires a stable current reference for excitation. This can be implemented in a number of ways, one which is shown in Figure 4. In this circuit, a voltage reference along with two operational amplifiers are used to generate a floating 1mA current source.

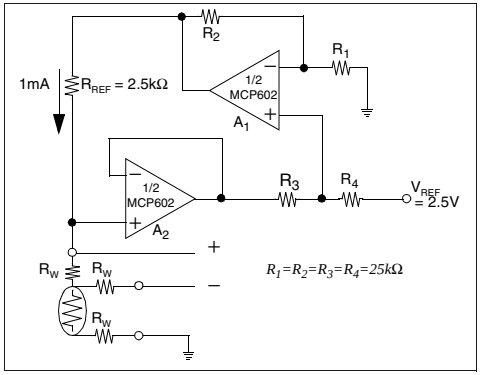


FIGURE 4: *A current source for the RTD element can be constructed in a single supply environment from two operational amplifiers and a precision voltage reference.*

This is accomplished by applying a 2.5V precision voltage reference to R_4 of the circuit. Since R_4 is equal to R_3 and the non-inverting input to A_1 is high impedance, the voltage drop across these two resistors is equal. The voltage between R_3 and R_4 is applied to the non-inverting input of A_1. That voltage is gained by $(1 + R_2/R_1)$ to the output of the amplifier and the top of the reference resistor, R_{REF}. If $R_1 = R_2$, the voltage at the output of A_1 is equal to:

$$V_{OUTA1} = (1 + R_2/R_1) \times (V_{REF} - V_{R4})$$
$$V_{OUTA1} = 2 \times (V_{REF} - V_{R4})$$

where

V_{OUTA1} is the voltage at the output of A_1 and

V_{R4} is the voltage drop across R_4.

The voltage at the output of A_2 is equal to:

$$V_{OUTA1} = V_{REF} - V_{R4} - V_{R3}$$

This same voltage appears at the inverting input of A_2 and across to the non-inverting input of A_2.

Solving these equations, the voltage drop across the reference resistor, R_{REF} is equal to:

$$V_{RREF} = V_{OUTA1} - V_{OUTA2}$$
$$V_{RREF} = 2 \times (V_{REF} - V_{R4}) - (V_{REF} - V_{R4} - V_{R3})$$
$$V_{RREF} = V_{REF}$$

where

V_{RREF} is the voltage across the reference resistor, R_{REF} and

V_{R3} is the voltage drop across R_3

The current through R_{REF} is equal to:

$$I_{RTD} = V_{REF} / R_{REF}$$

This circuit generates a current source that is ratiometric to the voltage reference. The same voltage reference can be used in other portions of the circuit, such as the analog-to-digital (A/D) converter reference.

Absolute errors in the circuit will occur as a consequence of the absolute voltage of the reference, the initial offset voltages of the operational amplifiers, the output swing of A_1, mismatches between the resistors, the absolute resistance value of R_{REF} and the RTD element. Errors due to temperature changes in the circuit will occur as a consequence of the temperature drift of the same elements listed above. The primary error sources over temperature are the voltage reference, offset drift of the operational amplifiers and the RTD element.

RTD SIGNAL CONDITIONING PATH

Changes in resistance of the RTD element over temperature is usually digitized through an A/D conversion as shown in Figure 5. The current excitation circuit shown in Figure 4 is used to excite the RTD element. With this style of excitation, the magnitude of the current source can be tuned to 1mA or less by adjusting R_{REF}. The voltage drop across the RTD element is sensed by A_3 then gained and filtered by A_4. With this circuit, a three-wire RTD element is selected. This configuration minimizes errors due to wire resistance and wire resistance drift over temperature.

In this circuit, the RTD element equals 100Ω at 0°C. If the RTD is used to sense temperature over its entire range of -200 to 600 °C, the range of resistance produced by the RTD would be nominally 23Ω to 331Ω.

Since the resistance range is relatively low, wire resistance and wire resistance change over temperature can skew the measurement of the RTD element. Consequently, a three wire RTD device is used to reduce these errors.

The errors contributed by the wire resistances, R_{W1} and R_{W3}, is subtracted from the circuit with the A_3 the operational amplifier circuit. In this configuration, R_1 and R_2 are equal and relatively high. The value of R_3 is selected to insure that the leakage currents through the resistor does not introduce errors to the current to the RTD element. The transfer function of this portion of the circuit is:

$$V_{OUTA3} = (V_{IN} - V_{w1})(1 + R_2/R_1) - V_{IN}(R_2/R_1)$$

where

$V_{IN} = V_{W1} + V_{RTD} + V_{W3}$,

V_{Wx} is the voltage drop across the wires to and from the RTD and

V_{OUTA3} is the voltage at the output of A_3.

If it is assumed that

$R_1 = R_2$ and $R_{W1} = R_{W3}$

the transfer function above reduces to:

$V_{OUTA3} = V_{RTD}$

The voltage signal at the output of A_3 is filtered with a 2nd order, low pass filter created with A_4, R_3, C_3, R_4, and C_4. This same signal is also gained by the resistors R_5 and R_6.

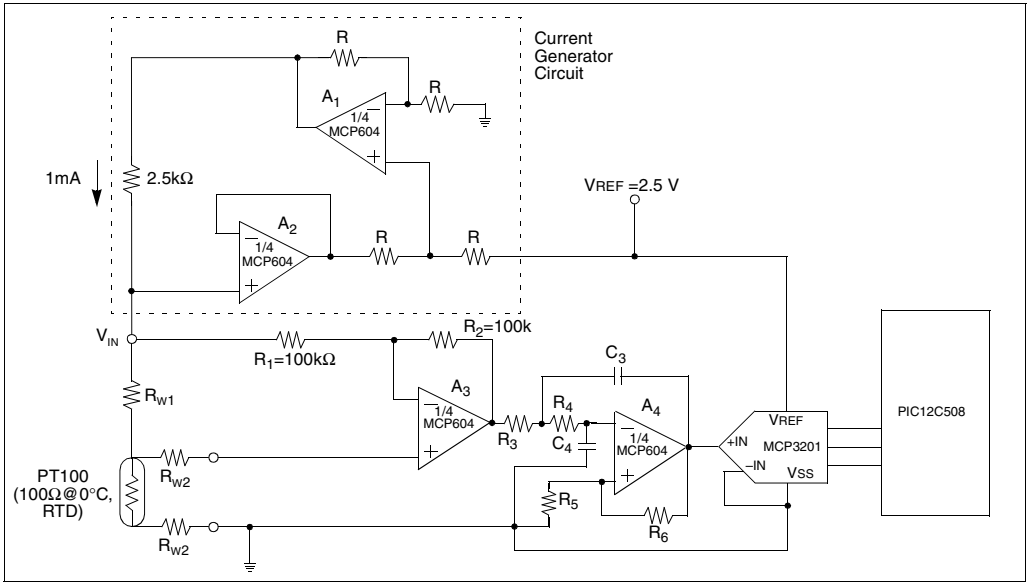


FIGURE 5: *This circuit uses an RTD temperature sensitive element to measure temperatures from -200 to 600°C. The current generator circuit from Figure 4 excites the sensor. An operational amplifier, A_3, is used to zero wire resistance error. A fourth amplifier, A_4 is used to gain the signal and filter possible alias interference. A 12-bit converter, MCP3201, converts the voltage across the RTD to digital code for the 8-pin controller, PIC12C508.*

CONCLUSION

Although the RTD requires a more circuitry in the signal conditioning path than the thermistor or the silicon temperature sensor, it ultimately provides a high precision, relatively linear result over a wider temperature range. If further linearization is performed in the processor, the RTD circuit can achieve ±0.01°C accuracy.

REFERENCES

Baker, Bonnie, “Temperature Sensing Technologies”, *AN679*, Microchip Technology Inc.

“Practical Temperature Measurements”, *OMEGA CATALOG*, pg Z-11

Baker, Bonnie, “Single Supply Temperature Sensing with Thermocouples”, *AN684*, Microchip Technology Inc.

Baker, Bonnie, “Using Operational Amplifiers for Analog Gain in Embedded System Design”, *AN682*, Microchip Technology Inc.

Baker, Bonnie, “Thermistors in Single Supply Temperature Sensing Circuits”, *AN685*, Microchip Technology Inc.

Hyde, Darrell, “Evaluating Thin Film RTD Stability”, *SENSORS*, OCT. 1997, pg 79

Madden, J.R., “Refresher on Resistance Temperature Devices”, *SENSORS*, Sept., 1997, pg 66

Li, Xumo, “Producing Higher Accuracy From SPRTs (Standard Platinum Resistance Thermometer)”, *MEASUREMENT & CONTROL,* June, 1996, pg118

AN688

Layout Tips for 12-Bit A/D Converter Application

Author: *Bonnie C. Baker*
Microchip Technology Inc.

INTRODUCTION

This Application Note originally started as a "cook book" for a true 12-bit layout. The assumption of this type of approach is that a reference design could be provided, which easily could be used for every layout implementation. But, the notion of this approach is fairly unrealistic. There are a multitude of successful ways to layout out systems with 12-bit Analog-to-Digital (A/D) Converters and each layout is highly dependent on the number of devices in the circuit, the types of the devices (digital or analog) and the environment that the final product will reside in. Given all of these variables, it could easily be demonstrated that one successful layout that provides twelve noise free bits from an analog signal may easily fail in another setting.

Because of the complexity of this problem, this Application Note will provide basic guidelines, ending with a review of issues to be aware of. Throughout the application note, examples of good layout and bad layout implementations will be presented. This will be done in the spirit of discussing concepts and not with the intent of recommending one layout as the only one to use.

GETTING A GOOD START

Imagine that the task at hand is to design a pressure sensing circuit that will accurately measure the pressure and present the results on an LCD display screen. Seems easy enough.

The circuit diagram for this system is shown in Figure 1. The pressure sensor that is chosen for the job is a piezo resistive sensor that is configured as a four element bridge. The particular sensor that is selected requires voltage excitation. The full swing output of the sensor is a small (10s of millivolts) differential signal that most appropriately is gained by an operational amplifier structure that also converts the differential output of the sensor to a single ended analog signal. A 12-bit converter is chosen to match the precision of the pressure sensor. Once the converter digitizes the voltage presented at its input, the digital code is sent to a microcontroller. The job of the microcontroller is to perform tasks such as calibration corrections and linearization. Once this is done, the results are sent to the LCD display.

The final step in the circuit development is to work through the calibration and linearization issues associated with the pressure sensor. Once these issues are settled, the microcontroller firmware is developed. Now the board is ready to go to layout.

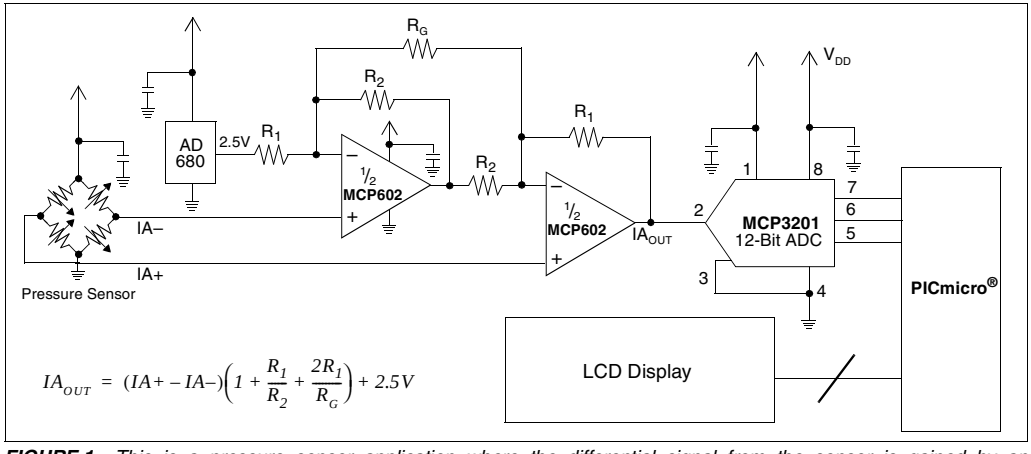


FIGURE 1: *This is a pressure sensor application where the differential signal from the sensor is gained by an instrumentation amplifier and digitized with a 12-bit A/D Converter, MCP3201. The results of the conversion is displayed on the LCD display.*

ONE MAJOR STEP TOWARDS DISASTER

The size of this circuit seems manageable. So small that one may be tempted to use an auto router layout tool. If this type of tool is used, it should be used carefully. If the tool is capable of implementing restrictions into the layout implementation, the layout design may have a fighting chance. If restrictions are not implemented by the auto routing tool, the best approach is to not use it at all.

GENERAL LAYOUT GUIDELINES

Device Placement

Device placement is critical. In general, there are some noise sensitive devices in this layout and other devices that are major problem creators. Here is a quick way to identify the good, from the bad, from the ugly.

1. Separate the circuit devices into two categories: high speed (>40MHz) and low speed.
2. Separate the above categories into three sub-categories: pure digital, pure analog, and mixed signal.

The board layout strategy should map the diagram shown in Figure 2. Notice the relationship of digital versus analog and high speed versus slower speeds to the board connector.

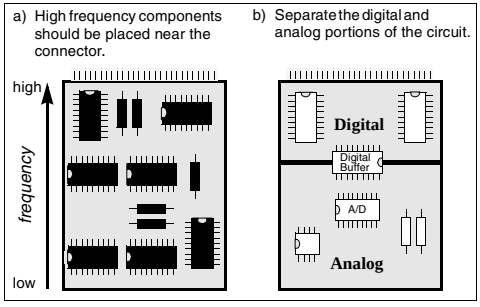


FIGURE 2: *The placement of an active component on a PCB is critical in precision 12-bit+ circuits.*

In Figure 2b. the digital and analog circuit is shown as being separate from the digital devices, which are closest to the connector or power supply.

The pure analog devices are furthest away for the digital devices to insure that switching noise is not coupled into the analog signal path.

The treatment of the A/D Converter in layout varies from technology to technology. For instance, if the A/D Converter uses a Successive Approximation Register (SAR) design approach, the entire device should be connected to the analog power and ground planes. A common error is to have the converter straddle the analog and digital planes. This strategy may work, but as the accuracy specifications of the A/D Converter improve the digital ground and power plane noise begins to cause problems. For high resolution SAR converters, a digital buffer should be used to isolate the converter from bus activity on the digital side of the circuit.

In contrast, if the A/D Converter is designed using a delta-sigma technology, it should straddle the analog and digital planes. This is due to the fact that the Delta-Sigma Converter is primarily a digital IC.

Ground and Power Supply Strategy

Once the general vicinity of the devices are determined, the ground planes and power planes should be defined. The strategy of the implementation of these planes are a bit tricky.

First of all, assuming that a ground plane is not needed is a dangerous assumption in any circuit with analog and/or mixed signal devices. Ground noise problems are more difficult to deal with than power supply noise problems because analog signals are most typically referenced to ground. For instance, in the circuit shown in Figure 1, the A/D Converter's inverting input pin (MCP3201) is connected to ground. Additionally, the negative side of the pressure sensor is also connected to ground.

A layout for the circuit in Figure 1 is shown in Figure 3. This layout implementation does not have ground or power planes on the board.

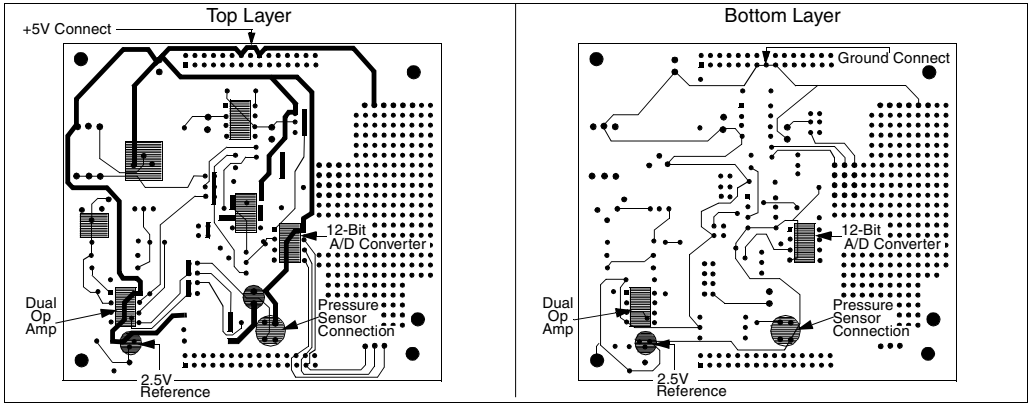


FIGURE 3: *Layout of the top and bottom layers of the circuit in Figure 1. Note that this layout does not have a ground or power plane.*

With this circuit layout, the controller is dedicated to interfacing with the converter and sending the converter's results to the LCD display. The digital output of the converter over time is shown in Figure 4. This data was collected with no excitation being applied to the sensor.

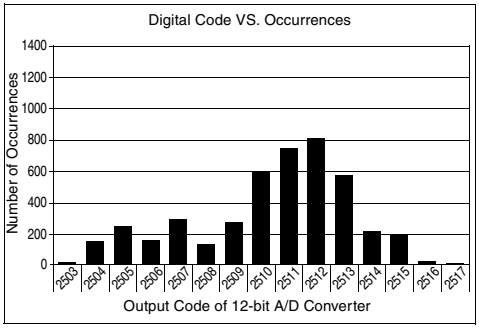


FIGURE 4: *This is a histogram of 4096 samples from the output of the A/D Converter from a PCB that does not have a ground or power plane as shown in the PCB layout in Figure 3. The by-pass capacitors are installed.*

When determining the grounding strategy of a board, the task at hand should actually be to determine if the circuit can work adequately with just one ground plane or does it need multiple planes.

Figure 5 shows the same layout shown in Figure 3, plus a ground plane. It should be noted that the ground plane has a few breaks due to signal traces. These breaks should be kept to a minimum. Current return paths should not be "pinched" as a consequence of these traces restricting the easy flow of current from the device to the power connector. The histogram for the A/D Converter output is shown in Figure 6. Compared to Figure 4, the output codes are much tighter. The same active devices were used for both tests. The passive devices were different causing a slight offset difference. The noise shown with the A/D Converter digital code is assignable to the op amp noise and the absence of an anti-aliasing filter.

If the circuit has a "minimum" amount of digital circuitry on board, a single ground plane and a single power plane may be appropriate. The qualifier "minimum" is defined by the board designer. The danger of connecting the digital and analog ground planes together is that the analog circuitry can pick-up the noise on the supply pins and couple it into the signal path. In either case, the analog and digital grounds and power supplies should be connected together at one or more points in the circuit to insure that the power supply, input and output ratings of all of the devices are not violated.

The inclusion of a power plane in a 12-bit system is not as critical as the required ground plane. Although a power plane can solve many problems, power noise can be reduced by making the power traces two or three times wider than other traces on the board and by using by-pass capacitors effectively.

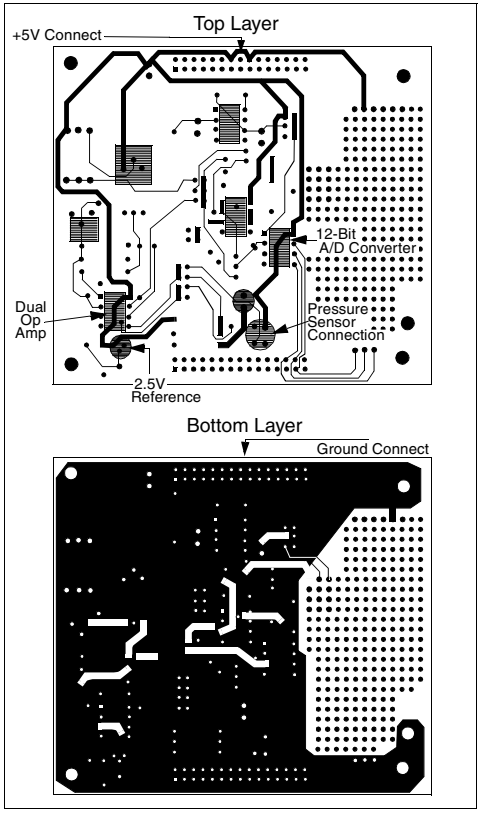


FIGURE 5: *Layout of the top and bottom layers of the circuit in Figure 1. Note that this layout DOES have a ground.*

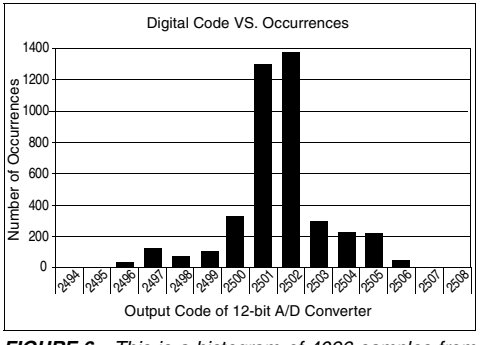


FIGURE 6: *This is a histogram of 4096 samples from the output of the A/D Converter on the PCB that has a ground plane as shown in the PCB layout in Figure 5. Note that the power traces are made considerably wider than the signal traces in order to reduce power supply trace inductance. This circuit has all by-pass capacitors installed.*

Signal Traces

Generally speaking, the signal traces on the board (both digital and analog) should be a short as possible. This basic guideline will minimize the opportunities for extraneous signals to couple into the signal path.

One area to be particularly cautious of is the input terminals of analog devices. These input terminals normally have a higher impedance than the output or power supply pins. As an example, the voltage reference input pin to the analog to digital converter is most sensitive while a conversion is occurring. With the type of 12-bit converter shown in Figure 1, the input terminals (IN+ and IN–) are also sensitive to injected noise.

Another potential for noise injection into the signal path is the input terminals of an operational amplifier. These terminals have typically 10^9 to 10^{13} Ω input impedance.

These high impedance input terminals are sensitive to injected currents. This can occur if the trace from a high impedance input is next to a trace that has fast changing voltages, such as a digital or clock signal. When a high impedance trace is in close proximity to a trace with these types of voltage changes, charge is capacitivly coupled into the high impedance trace.

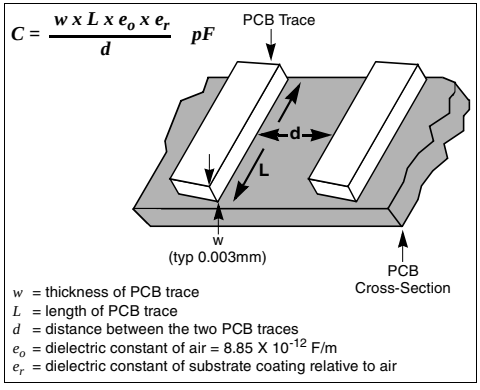


FIGURE 7: *A capacitor can be constructed on a PCB by placing two traces in close proximity. With this PCB capacitor, signals can be coupled between the traces.*

As shown in Figure 7, the value of the capacitance between two traces is primarily dependent on the distance (d) between the traces and the distance that the two traces are in parallel (L). From this model, the amount of current generated into the high impedance trace is equal to:

$$I = C\,\delta V/\delta t$$

where

I equals the current that appears on the high impedance trace

C equals the value of capacitance between the two PCB traces

δV equals the change in voltage of the trace that is switching, and

δt equals the amount of time that the voltage change took to get from one level to the next.

DID I SAY BY-PASS?

A good rule concerning by-pass capacitors is to always include them in the circuit. If they are not included, the power supply noise may very well eliminate any chance for 12-bit precision.

By-pass capacitors belong in two locations on the board: one at the power supply (10µF to 100µF or both) and one for every active device (digital and analog). The value of the device's by-pass capacitor is dependent on the device in question. If the bandwidth of the device is less than or equal to ~1MHz, a 1µF will reduce injected noise dramatically. If the bandwidth of the device is above ~10MHz, a 0.1µF capacitor is probably appropriate. In between these two frequencies, both or either one could be used. Refer to the manufacturer's guidelines for specifics.

Every active device on the board requires a by-pass capacitor. The by-pass capacitor must be placed as close as possible to the power supply pin of the device as shown in Figure 5. If two by-pass capacitors are used for one device, the smaller one should be closest to the device pin. Finally, the lead length of the by-pass capacitor should be as short as possible.

To illustrate the benefits of by-pass capacitors, data is collected from the layout shown in Figure 5, minus the by-pass capacitors. This data is shown in Figure 8.

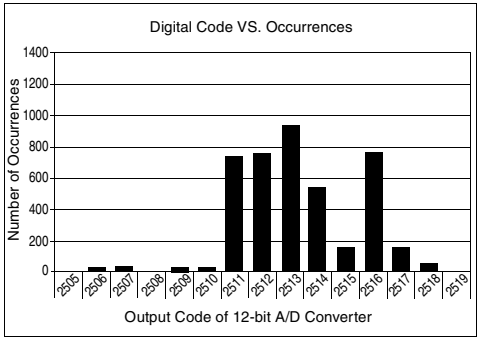


FIGURE 8: *This a histogram of 4096 samples from the output of the A/D Converter on the PCB that has a ground plane as shown in the PCB layout in Figure 3. With this circuit implementation, all by-pass capacitors have been removed.*

PCB DESIGN CHECK LIST

Good 12-bit layout techniques are not difficult to master as long as a few guidelines are considered.

1. Check device placement versus connectors. Make sure that high speed devices and digital devices are closest to the connector.
2. Always have at least one ground plane in the circuit.
3. Make power traces wider than other traces on the board.
4. Review current return paths and look for possible noise sources on ground connects. This is done by determining the current density at all points of the ground plane and the amount of possible noise present.
5. By-pass all devices properly. Place the capacitors as close to the power pins of the device as possible.
6. Keep all traces as short as possible.
7. Follow all high impedance traces looking for possible capacitive coupling problems from trace to trace.

REFERENCES

Morrison, Ralph, “Noise and Other Interfering Signals”, *John Wiley & Sons, Inc.*, 1992

Baker, Bonnie, “Noise Sources in Applications Using Capacitive Coupled Isolated Amplifiers”, *AB-047, Burr-Brown Corporation*

NOTES:

AN699

Anti-Aliasing, Analog Filters for Data Acquisition Systems

Author: *Bonnie C. Baker*
Microchip Technology Inc.

INTRODUCTION

Analog filters can be found in almost every electronic circuit. Audio systems use them for preamplification, equalization, and tone control. In communication systems, filters are used for tuning in specific frequencies and eliminating others. Digital signal processing systems use filters to prevent the aliasing of out-of-band noise and interference.

This application note investigates the design of analog filters that reduce the influence of extraneous noise in data acquisition systems. These types of systems primarily utilize low-pass filters, digital filters or a combination of both. With the analog low-pass filter, high frequency noise and interference can be removed from the signal path prior to the analog-to-digital (A/D) conversion. In this manner, the digital output code of the conversion does not contain undesirable aliased harmonic information. In contrast, a digital filter can be utilized to reduce in-band frequency noise by using averaging techniques.

Although the application note is about analog filters, the first section will compare the merits of an analog filtering strategy versus digital filtering.

Following this comparison, analog filter design parameters are defined. The frequency characteristics of a low pass filter will also be discussed with some reference to specific filter designs. In the third section, low pass filter designs will be discussed in depth.

The next portion of this application note will discuss techniques on how to determine the appropriate filter design parameters of an anti-aliasing filter. In this section, aliasing theory will be discussed. This will be followed by operational amplifier filter circuits. Examples of active and passive low pass filters will also be discussed. Finally, a 12-bit circuit design example will be given. All of the active analog filters discussed in this application note can be designed using Microchip's FilterLab software. FilterLab will calculate capacitor and resistor values, as well as, determine the number of poles that are required for the application. The program will also generate a SPICE macromodel, which can be used for spice simulations.

ANALOG VERSUS DIGITAL FILTERS

A system that includes an analog filter, a digital filter or both is shown in Figure 1. When an analog filter is implemented, it is done prior to the analog-to-digital conversion. In contrast, when a digital filter is implemented, it is done after the conversion from analog-to-digital has occurred. It is obvious why the two filters are implemented at these particular points, however, the ramifications of these restrictions are not quite so obvious.

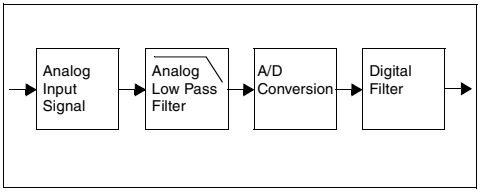


FIGURE 1: *The data acquisition system signal chain can utilize analog or digital filtering techniques or a combination of the two.*

There are a number of system differences when the filtering function is provided in the digital domain rather than the analog domain and the user should be aware of these.

Analog filtering can remove noise superimposed on the analog signal before it reaches the Analog-to-Digital Converter. In particular, this includes extraneous noise peaks. Digital filtering cannot eliminate these peaks riding on the analog signal. Consequently, noise peaks riding on signals near full scale have the potential to saturate the analog modulator of the A/D Converter. This is true even when the average value of the signal is within limits.

Additionally, analog filtering is more suitable for higher speed systems, i.e., above approximately 5kHz. In these types of systems, an analog filter can reduce noise in the out-of-band frequency region. This, in turn, reduces fold back signals (see the "Anti-Aliasing Filter Theory" section in this application note). The task of obtaining high resolution is placed on the A/D Converter. In contrast, a digital filter, by definition uses oversampling and averaging techniques to reduce in band and out of band noise. These two processes take time.

Since digital filtering occurs after the A/D conversion process, it can remove noise injected during the conversion process. Analog filtering cannot do this. Also,

the digital filter can be made programmable far more readily than an analog filter. Depending on the digital filter design, this gives the user the capability of programming the cutoff frequency and output data rates.

KEY LOW PASS ANALOG FILTER DESIGN PARAMETERS

A low pass analog filter can be specified with four parameters as shown in Figure 2 ($f_{CUT-OFF}$, f_{STOP}, A_{MAX}, and M).

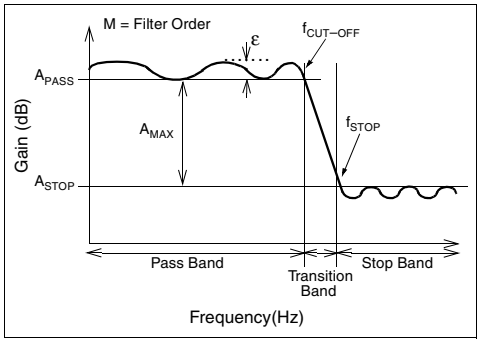


FIGURE 2: *The key analog filter design parameters include the –3dB cut-off frequency of the filter ($f_{cut-off}$), the frequency at which a minimum gain is acceptable (f_{stop}) and the number of poles (M) implemented with the filter.*

The cut-off frequency ($f_{CUT-OFF}$) of a low pass filter is defined as the -3dB point for a Butterworth and Bessel filter or the frequency at which the filter response leaves the error band for the Chebyshev.

The frequency span from DC to the cut-off frequency is defined as the pass band region. The magnitude of the response in the pass band is defined as A_{PASS} as shown in Figure 2. The response in the pass band can be flat with no ripple as is when a Butterworth or Bessel filter is designed. Conversely, a Chebyshev filter has a ripple up to the cut-off frequency. The magnitude of the ripple error of a filter is defined as ε.

By definition, a low pass filter passes lower frequencies up to the cut-off frequency and attenuates the higher frequencies that are above the cut-off frequency. An important parameter is the filter system gain, A_{MAX}. This is defined as the difference between the gain in the pass band region and the gain that is achieved in the stop band region or $A_{MAX} = A_{PASS} - A_{STOP}$.

In the case where a filter has ripple in the pass band, the gain of the pass band (A_{PASS}) is defined as the bottom of the ripple. The stop band frequency, f_{STOP}, is the frequency at which a minimum attenuation is reached. Although it is possible that the stop band has a ripple, the minimum gain (A_{STOP}) of this ripple is defined at the highest peak.

As the response of the filter goes beyond the cut-off frequency, it falls through the transition band to the stop band region. The bandwidth of the transition band is determined by the filter design (Butterworth, Bessel, Chebyshev, etc.) and the order (M) of the filter. The filter order is determined by the number of poles in the transfer function. For instance, if a filter has three poles in its transfer function, it can be described as a 3rd order filter.

Generally, the transition bandwidth will become smaller when more poles are used to implement the filter design. This is illustrated with a Butterworth filter in Figure 3. Ideally, a low-pass, anti-aliasing filter should perform with a “brick wall” style of response, where the transition band is designed to be as small as possible. Practically speaking, this may not be the best approach for an anti-aliasing solution. With active filter design, every two poles require an operational amplifier. For instance, if a 32nd order filter is designed, 16 operational amplifiers, 32 capacitors and up to 64 resistors would be required to implement the circuit. Additionally, each amplifier would contribute offset and noise errors into the pass band region of the response.

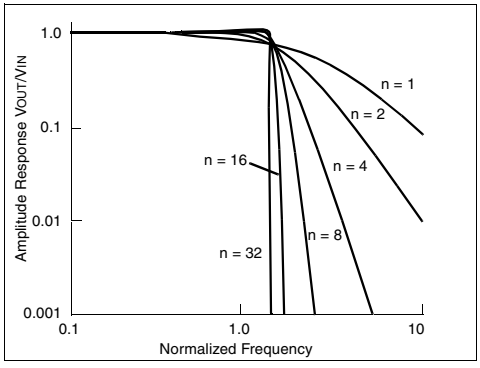


FIGURE 3: *A Butterworth design is used in a low pass filter implementation to obtain various responses with frequency dependent on the number of poles or order (M) of the filter.*

Strategies on how to work around these limitations will be discussed in the “Anti-Aliasing Theory” section of this application note.

ANALOG FILTER DESIGNS

The more popular filter designs are the Butterworth, Bessel, and Chebyshev. Each filter design can be identified by the four parameters illustrated in Figure 2. Other filter types not discussed in this application note include Inverse Chebyshev, Elliptic, and Cauer designs.

Butterworth Filter

The Butterworth filter is by far the most popular design used in circuits. The transfer function of a Butterworth filter consists of all poles and no zeros and is equated to:

$$V_{OUT}/V_{IN} = G/(a_0s^n + a_1s^{n-1} + a_2s^{n-2}... a_{n-1}s^2 + a_ns + 1)$$

where G is equal to the gain of the system.

Table 1 lists the denominator coefficients for a Butterworth design. Although the order of a Butterworth filter design theoretically can be infinite, this table only lists coefficients up to a 5th order filter.

M	a_0	a_1	a_2	a_3	a_4
2	1.0	1.4142136			
3	1.0	2.0	2.0		
4	1.0	2.6131259	3.4142136	2.6131259	
5	1.0	3.2360680	5.2360680	5.2360680	3.2360680

TABLE 1: *Coefficients versus filter order for Butterworth designs.*

As shown in Figure 4a., the frequency behavior has a maximally flat magnitude response in pass-band. The rate of attenuation in transition band is better than Bessel, but not as good as the Chebyshev filter. There is no ringing in stop band. The step response of the Butterworth is illustrated in Figure 5a. This filter type has some overshoot and ringing in the time domain, but less than the Chebyshev.

Chebyshev Filter

The transfer function of the Chebyshev filter is only similar to the Butterworth filter in that it has all poles and no zeros with a transfer function of:

$$V_{OUT}/V_{IN} = G/(a_0 + a_1s + a_2s^2+... a_{n-1}s^{n-1} + s_n)$$

Its frequency behavior has a ripple (Figure 4b.) in the pass-band that is determined by the specific placement of the poles in the circuit design. The magnitude of the ripple is defined in Figure 2 as ε. In general, an increase in ripple magnitude will lessen the width of the transition band.

The denominator coefficients of a 0.5dB ripple Chebyshev design are given in Table 2. Although the order of a Chebyshev filter design theoretically can be infinite, this table only lists coefficients up to a 5th order filter.

M	a_0	a_1	a_2	a_3	a_4
2	1.516203	1.425625			
3	0.715694	1.534895	1.252913		
4	0.379051	1.025455	1.716866	1.197386	
5	0.178923	0.752518	1.309575	1.937367	1.172491

TABLE 2: *Coefficients versus filter order for 1/2dB ripple Chebyshev designs.*

The rate of attenuation in the transition band is steeper than Butterworth and Bessel filters. For instance, a 5th order Butterworth response is required if it is to meet the transition band width of a 3rd order Chebyshev. Although there is ringing in the pass band region with this filter, the stop band is void of ringing. The step response (Figure 5b.) has a fair degree of overshoot and ringing.

Bessel Filter

Once again, the transfer function of the Bessel filter has only poles and no zeros. Where the Butterworth design is optimized for a maximally flat pass band response and the Chebyshev can be easily adjusted to minimize the transition bandwidth, the Bessel filter produces a constant time delay with respect to frequency over a large range of frequency. Mathematically, this relationship can be expressed as:

$$C = -\Delta\theta * \Delta f$$

where:

C is a constant,

θ is the phase in degrees, and

f is frequency in Hz

Alternatively, the relationship can be expressed in degrees per radian as:

$$C = -\Delta\theta / \Delta\omega$$

where:

C is a constant,

θ is the phase in degrees, and

ω is in radians.

The transfer function for the Bessel filter is:

$$V_{OUT}/V_{IN} = G/(a_0 + a_1s + a_2s^2+... a_{n-1}s^{n-1} + s_n)$$

The denominator coefficients for a Bessel filter are given in Table 3. Although the order of a Bessel filter design theoretically can be infinite, this table only lists coefficients up to a 5th order filter.

M	a_0	a_1	a_2	a_3	a_4
2	3	3			
3	15	15	6		
4	105	105	45	10	
5	945	945	420	105	15

TABLE 3: *Coefficients versus filter order for Bessel designs.*

The Bessel filter has a flat magnitude response in pass-band (Figure 4c). Following the pass band, the rate of attenuation in transition band is slower than the Butterworth or Chebyshev. And finally, there is no ringing in stop band. This filter has the best step response of all the filters mentioned above, with very little overshoot or ringing (Figure 5c.).

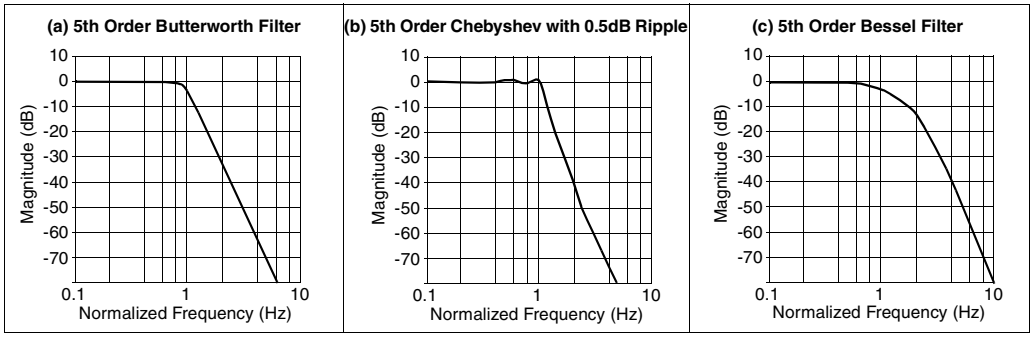


FIGURE 4: *The frequency responses of the more popular filters, Butterworth (a), Chebyshev (b), and Bessel (c).*

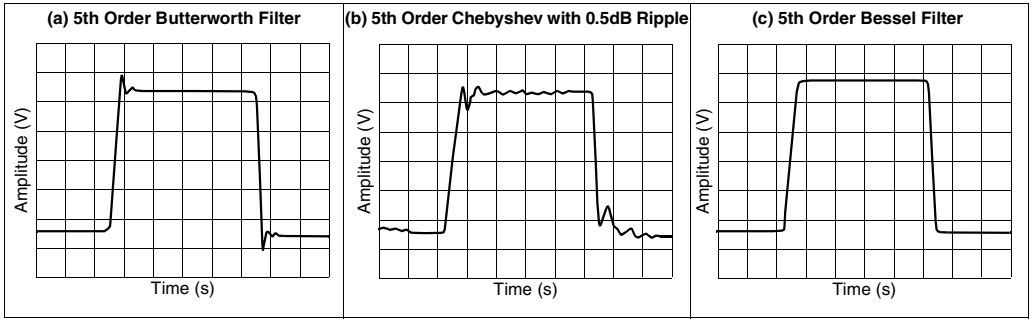


FIGURE 5: *The step response of the 5th order filters shown in Figure 4 are illustrated here.*

ANTI-ALIASING FILTER THEORY

A/D Converters are usually operated with a constant sampling frequency when digitizing analog signals. By using a sampling frequency (f_S), typically called the Nyquist rate, all input signals with frequencies below $f_S/2$ are reliably digitized. If there is a portion of the input signal that resides in the frequency domain above $f_S/2$, that portion will fold back into the bandwidth of interest with the amplitude preserved. The phenomena makes it impossible to discern the difference between a signal from the lower frequencies (below $f_S/2$) and higher frequencies (above $f_S/2$).

This aliasing or fold back phenomena is illustrated in the frequency domain in Figure 6.

In both parts of this figure, the x-axis identifies the frequency of the sampling system, f_S. In the left portion of Figure 6, five segments of the frequency band are identified. Segment N =0 spans from DC to one half of the sampling rate. In this bandwidth, the sampling system will reliably record the frequency content of an analog input signal. In the segments where N > 0, the frequency content of the analog signal will be recorded by the digitizing system in the bandwidth of the segment N = 0. Mathematically, these higher frequencies will be folded back with the following equation:

$$f_{ALIASED} = |f_{IN} - Nf_S|$$

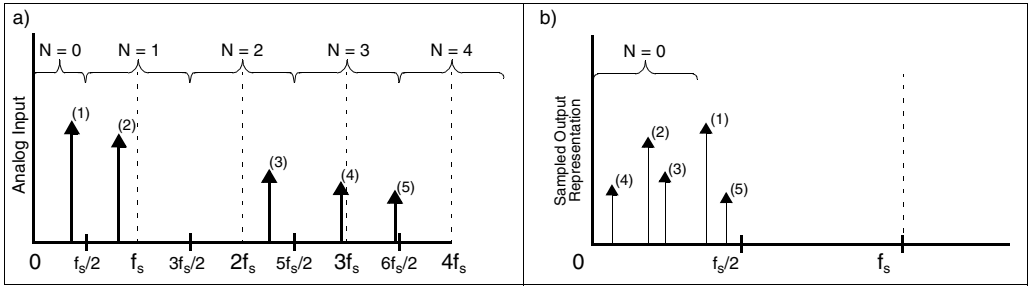


FIGURE 6: *A system that is sampling an input signal at f_s (a) will identify signals with frequencies below $f_s/2$ as well as above. Input signals below $f_s/2$ will be reliably digitized while signals above $f_s/2$ will be folded back (b) and appear as lower frequencies in the digital output.*

For example, let the sampling rate, (f_S), of the system be equal to 100kHz and the frequency content of:

$f_{IN}(1) = 41kHz$

$f_{IN}(2) = 82kHz$

$f_{IN}(3) = 219kHz$

$f_{IN}(4) = 294kHz$

$f_{IN}(5) = 347kHz$

The sampled output will contain accurate amplitude information of all of these input signals, however, four of them will be folded back into the frequency range of DC to $f_S/2$ or DC to 50kHz. By using the equation $f_{OUT} = |f_{IN} - Nf_S|$, the frequencies of the input signals are transformed to:

$f_{OUT}(1) = |41kHz - 0 \times 100kHz| = 41kHz$

$f_{OUT}(2) = |82kHz - 1 \times 100kHz| = 18kHz$

$f_{OUT}(3) = |219kHz - 2 \times 100kHz| = 19kHz$

$f_{OUT}(4) = |294kHz - 3 \times 100kHz| = 6kHz$

$f_{OUT}(5) = |347kHz - 4 \times 100kHz| = 53kHz$

Note that all of these signal frequencies are between DC and $f_S/2$ and that the amplitude information has been reliably retained.

This frequency folding phenomena can be eliminated or significantly reduced by using an analog low pass filter prior to the A/D Converter input. This concept is illustrated in Figure 7. In this diagram, the low pass filter attenuates the second portion of the input signal at frequency (2). Consequently, this signal will not be aliased into the final sampled output. There are two regions of the analog low pass filter illustrated in Figure 7. The region to the left is within the bandwidth of DC to $f_S/2$. The second region, which is shaded, illustrates the transition band of the filter. Since this region is greater than $f_S/2$, signals within this frequency band will be aliased into the output of the sampling system. The affects of this error can be minimized by moving the corner frequency of the filter lower than $f_S/2$ or increasing the order of the filter. In both cases, the minimum gain of the filter, A_{STOP}, at $f_S/2$ should less than the signal-to-noise ratio (SNR) of the sampling system.

For instance, if a 12-bit A/D Converter is used, the ideal SNR is 74dB. The filter should be designed so that its gain at f_{STOP} is at least 74dB less than the pass band gain. Assuming a 5th order filter is used in this example:

$f_{CUT\text{-}OFF} = 0.18f_S/2$ *for a Butterworth Filter*

$f_{CUT\text{-}OFF} = 0.11f_S/2$ *for a Bessel Filter*

$f_{CUT\text{-}OFF} = 0.21f_S/2$ *for a Chebyshev Filter with 0.5dB ripple in the pass band*

$f_{CUT\text{-}OFF} = 0.26f_S/2$ *for a Chebyshev Filter with 1dB ripple in the pass band*

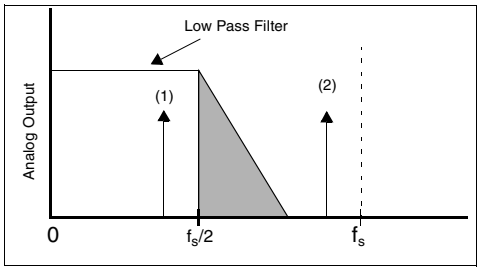


FIGURE 7: *If the sampling system has a low pass analog filter prior to the sampling mechanism, high frequency signals will be attenuated and not sampled.*

ANALOG FILTER REALIZATION

Traditionally, low pass filters were implemented with passive devices, ie. resistors and capacitors. Inductors were added when high pass or band pass filters were needed. At the time active filter designs were realizable, however, the cost of operational amplifiers was prohibitive. Passive filters are still used with filter design when a single pole filter is required or where the bandwidth of the filter operates at higher frequencies than leading edge operational amplifiers. Even with these two exceptions, filter realization is predominately implemented with operational amplifiers, capacitors and resistors.

Passive Filters

Passive, low pass filters are realized with resistors and capacitors. The realization of single and double pole low pass filters are shown in Figure 8.

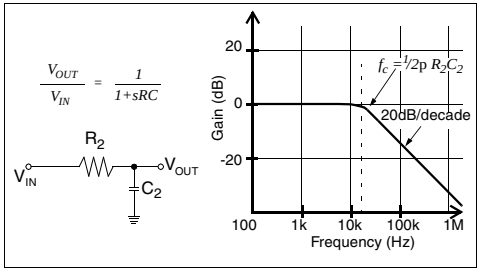


FIGURE 8: *A resistor and capacitor can be used to implement a passive, low pass analog filter. The input and output impedance of this type of filter implementation is equal to R_2.*

The output impedance of a passive low pass filter is relatively high when compared to the active filter realization. For instance, a 1kHz low pass filter which uses a 0.1µF capacitor in the design would require a 1.59kΩ resistor to complete the implementation. This value of resistor could create an undesirable voltage drop or make impedance matching difficult. Consequently, passive filters are typically used to implement a single pole. Single pole operational amplifier filters have the added benefit of "isolating" the high impedance of the filter from the following circuitry.

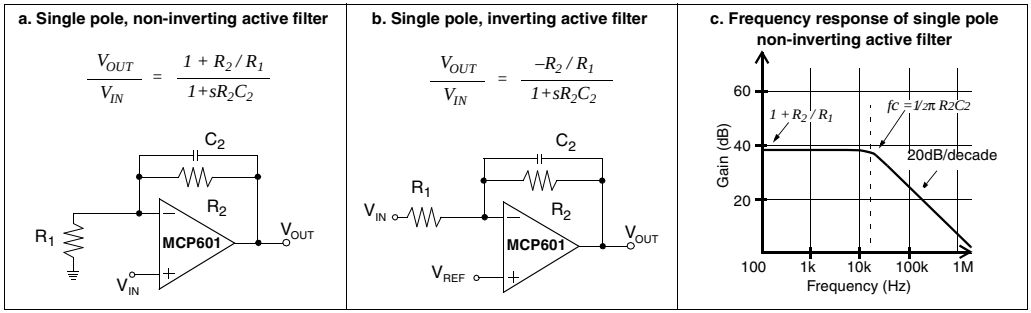


FIGURE 9: *An operational amplifier in combination with two resistors and one capacitor can be used to implement a 1st order filter. The frequency response of these active filters is equivalent to a single pole passive low pass filter.*

It is very common to use a single pole, low pass, passive filter at the input of a Delta-Sigma A/D Converter. In this case, the high output impedance of the filter does not interfere with the conversion process.

Active Filters

An active filter uses a combination of one amplifier, one to three resistors and one to two capacitors to implement one or two poles. The active filter offers the advantage of providing "isolation" between stages. This is possible by taking advantage of the high input impedance and low output impedance of the operational amplifier. In all cases, the order of the filter is determined by the number of capacitors at the input and in the feedback loop of the amplifier.

Single Pole Filter

The frequency response of the single pole, active filter is identical to a single pole passive filter. Examples of the realization of single pole active filters are shown in Figure 9.

Double Pole, Voltage Controlled Voltage Source

The Double Pole, Voltage Controlled Voltage Source is better know as the Sallen-Key filter realization. This filter is configured so the DC gain is positive. In the Sallen-Key Filter realization shown in Figure 10, the DC gain is greater than one. In the realization shown in Figure 11, the DC gain is equal to one. In both cases, the order of the filters are equal to two. The poles of these filters are determined by the resistive and capacitive values of R_1, R_2, C_1 and C_2.

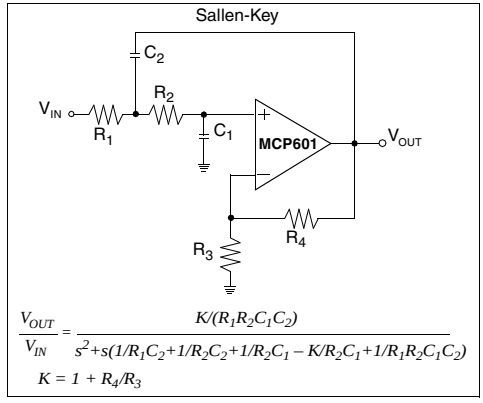


FIGURE 10: *The double pole or Sallen-Key filter implementation has a gain G = 1 + R_4/R_3. If R_3 is open and R_4 is shorted the DC gain is equal to 1 V/V.*

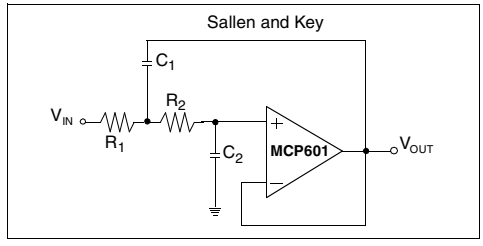
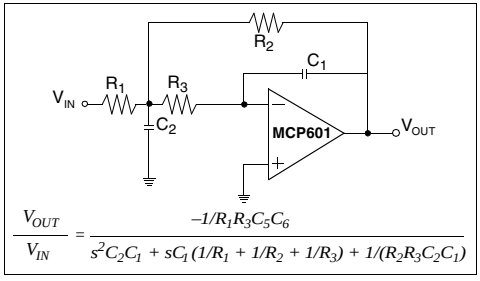


FIGURE 11: *The double pole or Sallen-Key filter implementation with a DC gain is equal to 1V/V.*

Double Pole Multiple Feedback

The double pole, multiple feedback realization of a 2nd order low pass filter is shown in Figure 12. This filter can also be identified as simply a Multiple Feedback Filter. The DC gain of this filter inverts the signal and is equal to the ratio of R_1 and R_2. The poles are determined by the values of R_1, R_3, C_1, and C_2.



$$\frac{V_{OUT}}{V_{IN}} = \frac{-1/R_1R_3C_5C_6}{s^2C_2C_1 + sC_1(1/R_1 + 1/R_2 + 1/R_3) + 1/(R_2R_3C_2C_1)}$$

FIGURE 12: *A double pole, multiple feedback circuit implementation uses three resistors and two capacitors to implement a 2nd order analog filter. DC gain is equal to $-R_2 / R_1$.*

ANTI-ALIASING FILTER DESIGN EXAMPLE

In the following examples, the data acquisition system signal chain shown in Figure 1 will be modified as follows. The analog signal will go directly into an active low pass filter. In this example, the bandwidth of interest of the analog signal is DC to 1kHz. The low pass filter will be designed so that high frequency signals from the analog input do not pass through to the A/D Converter in an attempt to eliminate aliasing errors. The implementation and order of this filter will be modified according to the design parameters. Excluding the filtering function, the anti-aliasing filter will not modify the signal further, i.e., implement a gain or invert the signal. The low pass filter segment will be followed by a 12-bit SAR A/D Converter. The sampling rate of the A/D Converter will be 20kHz, making 1/2 of Nyquist equal to 10kHz. The ideal signal-to-noise ratio of a 12-bit A/D Converter of 74dB. This design parameter will be used when determining the order of the anti-aliasing filter. The filter examples discussed in this section were generated using Microchip's FilterLab software.

Three design parameters will be used to implement appropriate anti-aliasing filters:

1. Cut-off frequency for filter must be 1kHz or higher.
2. Filter attenuates the signal to -74dB at 10kHz.
3. The analog signal will only be filtered and not gained or inverted.

Implementation with Bessel Filter Design

A Bessel Filter design is used in Figure 13 to implement the anti-aliasing filter in the system described above. A 5th order filter that has a cut-off frequency of 1kHz is required for this implementation. A combination of two Sallen-Key filters plus a passive low pass filter are designed into the circuit as shown in Figure 14. This filter attenuates the analog input signal 79dB from the pass band region to 10kHz. The frequency response of this Bessel, 5th order filter is shown in Figure 13.

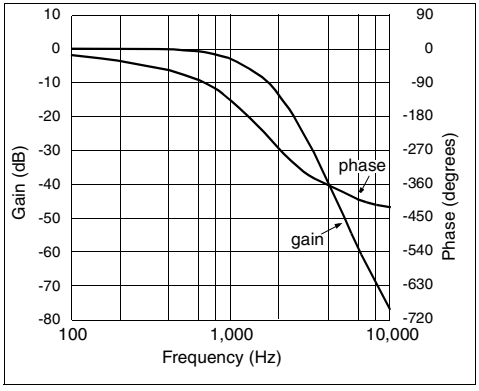


FIGURE 13: *Frequency response of 5th order Bessel design implemented in Figure 14.*

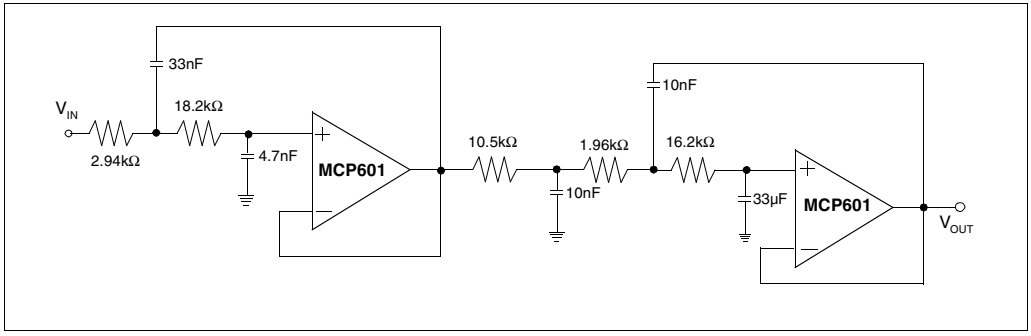


FIGURE 14: *5th order Bessel design implemented two Sallen-Key filters and on passive filter. This filter is designed to be an anti-aliasing filter that has a cut-off frequency of 1kHz and a stop band frequency of ~5kHz.*

Implementation with Chebyshev Design

When a Chebyshev filter design is used to implement the anti-aliasing filter in the system described above, a 3rd order filter is required, as shown Figure 15.

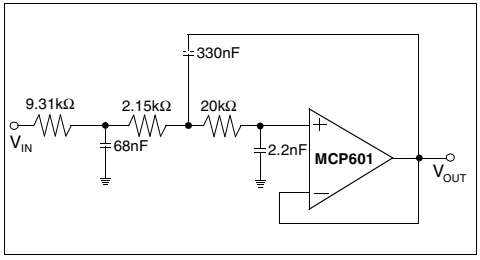


FIGURE 15: *3rd order Chebyshev design implemented using one Sallen-Key filter and one passive filter. This filter is designed to be an anti-aliasing filter that has a cut-off frequency of 1kHz -4db ripple and a stop band frequency of ~5kHz.*

Although the order of this filter is less than the Bessel, it has a 4dB ripple in the pass band portion of the frequency response. The combination of one Sallen-Key filter plus a passive low pass filter is used. This filter is attenuated to -70dB at 10kHz. The frequency response of this Chebyshev 3rd order filter is shown in Figure 16.

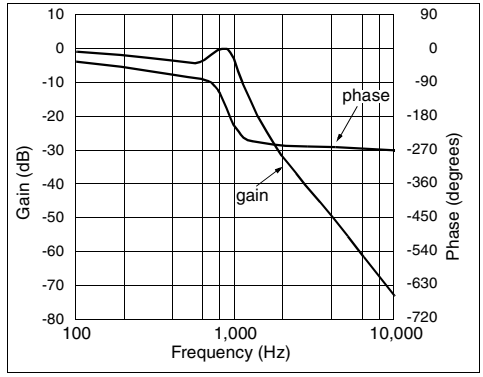


FIGURE 16: *Frequency response of 3rd order Chebyshev design implemented in Figure 15.*

This filter provides less than the ideal 74dB of dynamic range (A_{MAX}), which should be taken into consideration.

The difference between -70dB and -74dB attenuation in a 12-bit system will introduce little less than 1/2 LSB error. This occurs as a result of aliased signals from 10kHz to 11.8KHz. Additionally, a 4dB gain error will occur in the pass band. This is a consequence of the ripple response in the pass band, as shown in Figure 16.

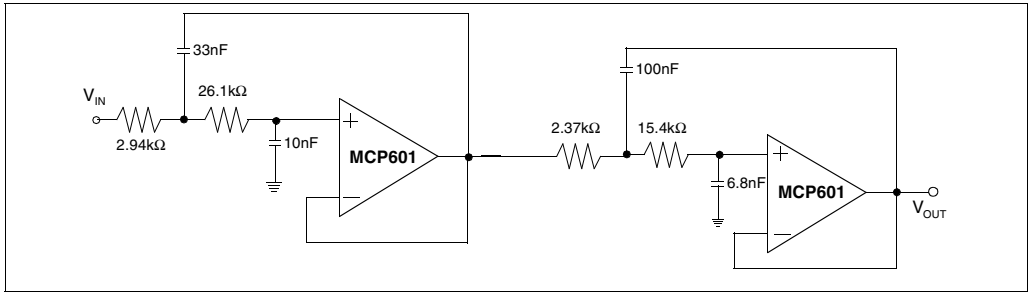


FIGURE 17: *4th order Butterworth design implemented two Sallen-Key filters. This filter is designed to be an anti-aliasing filter that has a cut-off frequency of 1kHz and a stop band frequency of ~5kHz.*

Implementation with Butterworth Design

As a final alternative, a Butterworth filter design can be used in the filter implementation of the anti-aliasing filter, as shown in Figure 17.

For this circuit implementation, a 4th order filter is used with a cut-off frequency of 1kHz. Two Sallen-Key filters are used. This filter attenuates the pass band signal 80dB at 10kHz. The frequency response of this Butterworth 4th order filter is shown in Figure 18.

The frequency response of the three filters described above along with several other options are summarized in Table 4.

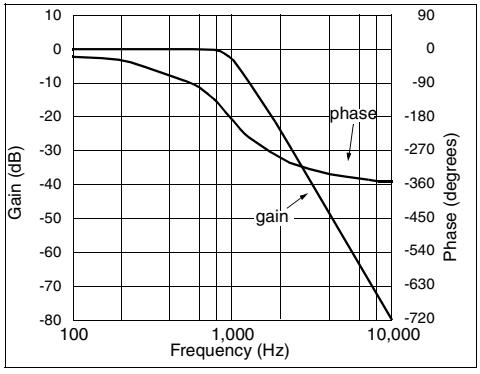


FIGURE 18: *Frequency response of 4th order Butterworth design implemented in Figure 17.*

FILTER ORDER, M	BUTTERWORTH, A_{MAX} (dB)	BESSEL, A_{MAX} (dB)	CHEBYSHEV, A_{MAX} (dB) W/ RIPPLE ERROR OF 1dB	CHEBYSHEV, A_{MAX} (dB) W/ RIPPLE ERROR OF 4dB
3	60	51	65	70
4	80	66	90	92
5	100	79	117	122
6	120	92	142	144
7	140	104	169	174

TABLE 4: *Theoretical frequency response at 10kHz of various filter designs versus filter order. Each filter has a cut-off frequency of 1kHz.*

CONCLUSION

Analog filtering is a critical portion of the data acquisition system. If an analog filter is not used, signals outside half of the sampling bandwidth of the A/D Converter are aliased back into the signal path. Once a signal is aliased during the digitalization process, it is impossible to differentiate between noise with frequencies in band and out of band.

This application note discusses techniques on how to determine and implement the appropriate analog filter design parameters of an anti-aliasing filter.

REFERENCES

Baker, Bonnie, “Using Operational Amplifiers for Analog Gain in Embedded System Design”, *AN682, Microchip Technologies, Inc.*

Analog Filter Design, Valkenburg, M. E. Van, *Oxford University Press.*

Active and Passive Analog Filter Design, An Introduction, Huelsman, Lawrence p., *McGraw Hill, Inc.*

AN702

Interfacing Microchip MCP3201 A/D Converter to 8051-Based Microcontroller

Author: *Lee Studley*
Microchip Technology Inc.

INTRODUCTION

In embedded controller applications, it is often desirable to provide a means to digitize analog signals. The MCP3201 12-bit Analog-to-Digital (A/D) Converter gives the designer an easy means to add this feature to a microcontroller with a minimal number of connections.

This Application Note will demonstrate how easy it is to connect the MPC3201 to an 8051-compatible microprocessor.

The MCP3201 is a fast 100kHz 12-bit A/D Converter featuring low power consumption and power saving standby modes. The features of the device include an onboard sample-hold and a single pseudo differential input. Output data from the MCP3201 is provided by a high speed serial interface that is compatible with the SPI® protocol. The MCP3201 operates over a broad voltage range (2.7V – 5.5V). The device is offered in 8-pin PDIP and 150mil SOIC packages.

The MCP3201 connects to the target microprocessor via an SPI-like serial interface that can be controlled by I/O commands, or by using the synchronous resources commonly found in microcontrollers. Two methods will be explored in supporting the serial format for the A/D Converter: An I/O port "bit-banging" method and a method that uses the 8051 UART in synchronous serial mode 0. An 8051 derivative processor, the 80C320, was chosen for testing since it has a second onboard serial port. This second serial port allows the A/D Converter sample data to be echoed to a host PC running an ASCII terminal program such as Hyperterm. Both ports respond to the standard 8051 setup instructions for code portability. An 8051 has a single UART that can be dedicated to either the A/D Converter, or to other communication tasks.

I/O PORT METHOD

The serial data format supported by the MCP3201 is illustrated in Figure 1. The A/D Converter will come out of its sleep mode on the falling edge of $\overline{CS}$. The conversion is then initiated with the first rising edge of CLK. During the next 1.5 CLK cycles, the converter samples the input signal. The sampling period stops at the end of the 1.5 CLK cycles on the falling edge of CLK, and D_{OUT} also changes from a Hi-Z state to null. Following the transmission of the null bit, the A/D Converter will respond by shifting out conversion data on each subsequent falling edge of the clock. The most significant bits are clocked out first. The micro is supplying the $\overline{CS}$ and CLK signals and the A/D Converter responds with the bit data on D_{OUT}.

As shown in Figure 1, starting with an initial NULL bit, bits B11, B10, B9…B0 are shifted out of the A/D Converter. Following bit B0, further CLK falling edges will cause the A/D Converter to shift out bits B1…B11 in reverse order of the initial bit sequence. Continued CLKs will shift out zeros following B11 until $\overline{CS}$ returns high to signal the end of the conversion. On the rising edge of $\overline{CS}$, D_{OUT} will change to a Hi-Z state. The device receiving the data from the A/D Converter can use the low-to-high edge of CLK to validate (or latch) the A/D Converter bit data at D_{OUT}.

The 8051 instruction set provides for bit manipulation to allow the use of I/O pins to serve as a serial host for the A/D Converter. By manually toggling the I/O pins and reading the resulting A/D Converter D_{OUT} bits, the designer is free to use any I/O pin that can provide the needed function. The drawback to this method is the bandwidth limit imposed by the execution time of the opcodes supporting the A/D Converter communication. Example 1 shows a code module for a simple I/O port "bit-banging" method for supporting the MCP3201. To optimize for speed, the result is right justified in the ADRESH:ADRESL register pair.

SPI is a registered trademark of Motorola

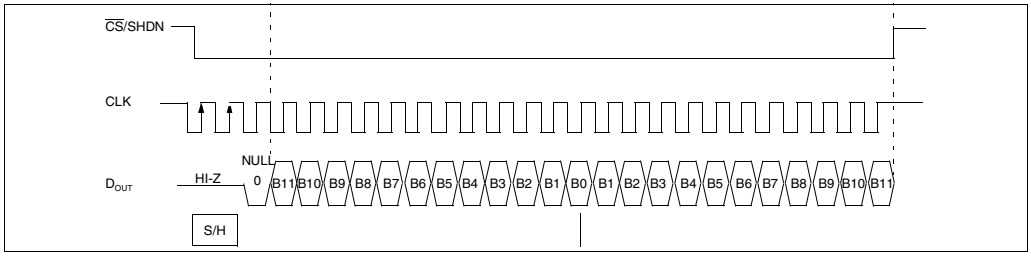


FIGURE 1: *MCP3201 Serial Data Format.*

EXAMPLE 1: I/O PORT METHOD CODE

```
GET_AD:     SETB CS                 ; set cs hi
            MOV COUNTA,#15          ;

NXTBIT:     CLR DCLK                ; X,X,NULL,D11,D10,D9...D0
            CLR CS                  ; CS low to start conversion or keep low till done
            SETB DCLK               ; raise the clock
            MOV C,SDAT              ; put data into C flag
            RLC A                   ; shift C into Acc (A/D low bits)
            XCH A,ADRESH            ; get ADRESH byte (save low bits in ADRESH for now)
            RLC A                   ; shift C into Acc (A.D high bits)
            XCH A,ADRESH            ; get low bits back into Acc for next loop
            DJNZ COUNTA,NXTBIT
            MOV ADRESL,A            ; put A into ADRESL
            ANL ADRESH,#0FH         ; mask off unwanted bits (x,X,X,Null)
            SETB CS                 ; set CS hi to end conversion
```

USING THE SERIAL PORT IN SYNCHRONOUS MODE0

The UART on the 8051 supports a synchronous shift register mode that, with some software help, can be used to speed up the communications to the A/D Converter. In Mode0, the UART uses the RX pin for data I/O, while the TX pin provides a synchronization clock. The shift register is 8 bits wide and the TX pin will transition low to high to supply a clock rising edge for each bit. Figure 2 shows the typical Mode0 timing.

Since the UART was designed primarily to support RS-232 data transfers, the bit order expected is LSb first. The shift register Mode0 also uses this bit order. As shown in Figure 1, the first 12 bits of the A/D Converter data are 'backwards' for our application. Fortunately, the MCP3201 provides the reverse order of sampled bits after the initial transfer of bits B11…B0.

Inspection of Figure 1 readily shows that working back from the last data bit transferred, 3 bytes received from the shift register will cover 24 bits of the 26 bits transferred from the A/D Converter. Conveniently, bit manipulation can be used to provide the two CLK rising edges needed during the beginning sample operation. After these two initial CLK cycles, the UART shifter can be accessed three times to read in the remainder of the data. The bit order will be correct for the third shifter byte as MSB data, the second byte will have 4 LSBs in the upper nibble (the lower nibble will be masked off), and the first byte will be tossed. Figure 3 shows the relationship between the shifted bits and SBUF data received by the UART. Example 2 shows a code module for using the synchronous port as the interface. The result is left justified in the ADRESH:ADRESL register pair.

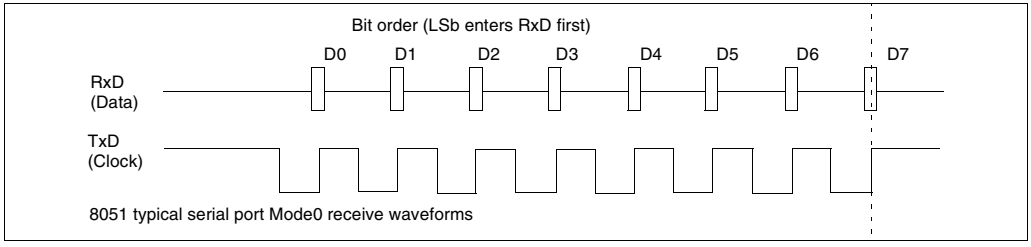


FIGURE 2: *Typical 8051 UART Mode0 Timing.*

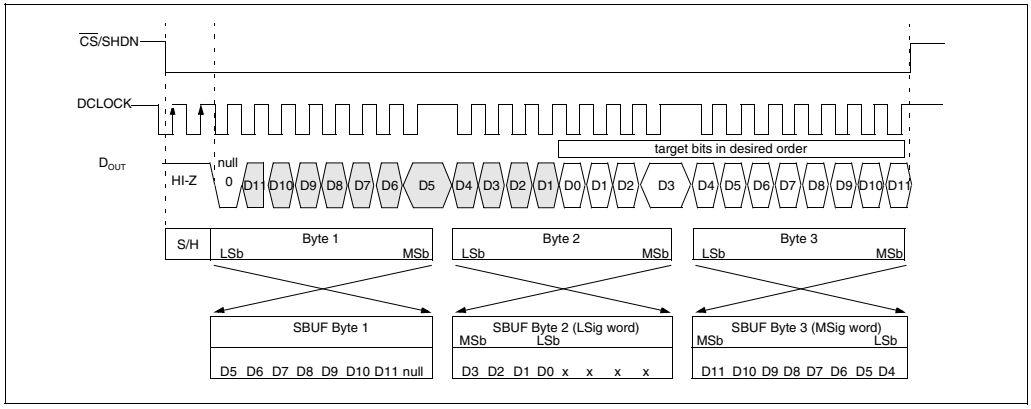


FIGURE 3: *Serial Port Waveforms.*

EXAMPLE 2: SYCHRONOUS PORT CODE

```
GET_AD:     SETB   CS                ; set CS hi
            CLR    DCLK              ; X,X,NULL,D11,D10,D9...D0
            CLR    CS                ; CS low to start conversion or keep low till done
            SETB   DCLK              ; 1st S/H clock
            CLR    DCLK              ;
            SETB   DCLK              ; 2nd S/H clock and leave DCLK high

            SETB   REN_1             ; REN=1 & R1_1=0 initiates a receive
            CLR    R1_1              ;

BYTE_1:     JNB    R1_1,BYTE_1
            MOV    A,SBUF1           ; toss this byte
            CLR    R1_1
BYTE_2:     JNB    R1_1,BYTE_2
            MOV    ADRESL,SBUF1      ; save LSbs
            CLR    R1_1
BYTE_3:     JNB    R1_1,BYTE_3
            MOV    ADRESH,SBUF1      ; save MSbs
            SETB   CS                ; set CS hi to end conversion
            ANL    ADRESL,#0FH       ; mask off unwanted LSb bits
```

A Quick Comparison of Results

The test circuit used was taken from the data sheet and is shown in Figure 4.

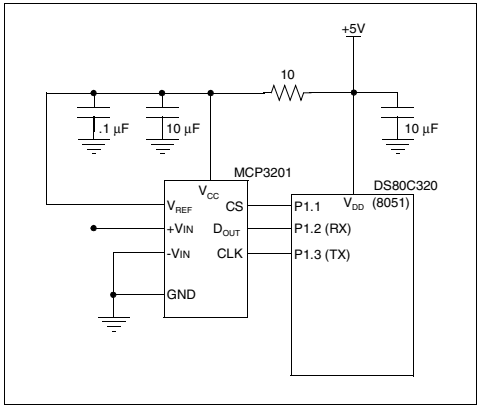


FIGURE 4: *Test Circuit.*

Oscilloscope screen shots of the I/O port method vs. the Synchronous Port method are shown in Figure 5 and Figure 6.

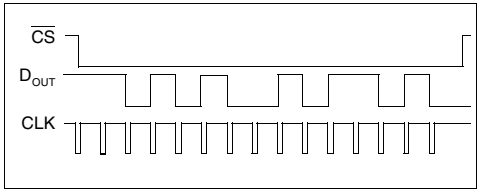


FIGURE 5: *Scope Shot: I/O Port Method.*

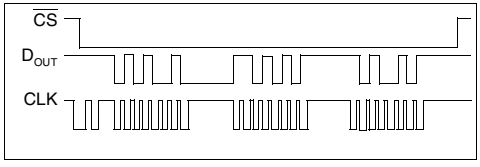


FIGURE 6: *Scope Shot: Synchronous Port Method.*

An 80C320 microprocessor clocked at a crystal frequency of 11.0592 MHz yielded the following results:

Method	CS Time (Conv. time approx.)	Approx. Throughput	Resources Used
I/O Port	99 µs	10 kHz	3 I/O pins (P1.1..P1.3)
Sync. Serial	43.4 µs	23 kHz	3 I/O pins (P1.1..P1.3) 1 UART (Mode0)

Note: The 80C320 can be clocked to 33MHz, which would effectively decrease the conversion time by a factor of 3 for increased performance in demanding applications.

TABLE 1: *Conversion Time Comparison.*

IN SUMMARY

Both methods illustrate the ease with which the MCP3201 A/D Converter can complement a design to add functionality for processing analog signals. The synchronous serial port method provides a 2:1 performance increase over the I/O port method, but consumes one UART as a resource. The I/O port method is flexible in allowing any suitable 3 I/O pins to be used in the interface.

Potential applications include control voltage monitoring, data logging, and audio processing. The routines in the source code appendices provide the designer with an effective resource to implement the design.

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX A: I/O PORT SOURCE CODE

```
;
;
$MOD51
$TITLE(ads)
$DATE(7/19/98)
$PAGEWIDTH(132)
$OBJECT(C:\ASM51\ads.OBJ)
;
; Author Lee Studley
; Assembled with Metalink's FreeWare ASM51 assembler
; Tested with NOICE emulation software.
; Tested with a DALLAS DS80C320 (8031) micro clocked @ 11.0592mhz
; This test uses a 'bit banging' approach yielding a conversion time
; of approximately 99uS
; The result is transmitted via the original 8051 UART to an ascii
; terminal at 19.2k baud 8N1 format
;
;================= RESET AND INTERRUPT VECTORS ======================
          ;
RSTVEC EQU 0000H;
IE0VEC EQU 0003H;
TF0VEC EQU 000BH;
IE1VEC EQU 0013H;
TF1VEC EQU 001BH;
RITIVEC EQU 0023;
TF2VEC EQU 002BH;( 8052 )
          ;
;================= VARIABLES ================
          DSEG

;================= PROGRAM VARIABLES ================
COUNTA    EQU 30H
COUNTB    EQU 31H
ADRESL    EQU 2
ADRESH    EQU 3

;================= HARDWARE EQUATES =================
DCLK      EQU P1.3
SDAT      EQU P1.2
CS        EQU P1.1

;
;================= CONSTANTS =================
;
;================= PROGRAM CODE =================
;
CSEG

;org RSTVEC
;LJMP START

ORG 4000H ; NOICE SRAM/PROGRAM SPACE

;=====================================================================
START:
;=====================================================================
;  Initialize the on-chip serial port for mode 1
;  Set timer 1 for baud rate: auto reload timer
;=====================================================================
SETUPUART:
        MOV   PCON,#80H; SET FOR DOUBLE BAUD RATE
        MOV   TMOD,#00100010B; two 8-bit auto-reload counters
```

```
        MOV   TH1, #0FDH; 19.2K @ 11.059 MHZ
        MOV   SCON,#01010010B; mode 1, TI set
        SETB  TR1; start timer for serial port
;=====================================================================
; GET_AD: Initiates the A/D conversion and retreives the AD sample into
; ADRESH,ADRESL.
; The A/D convertor is connected to port1 pins 0..2 as:
;         SDAT EQU P1.0  I/O
;         DCLK EQU P1.1  I/O
;         CS   EQU P1.2  I/O
; Uses: ADRESL,ADRESH,ACC,COUNTA
; Exits: ADRESH=(x,x,x,x,B11..B8), ADRESL(B7..B0,)
;=====================================================================

GET_AD:   SETB CS                  ; set cs hi
          MOV COUNTA,#15           ; number of bits to shift 12+X,X,NULL=15

NXTBIT:   CLR DCLK                 ; X,X,NULL,D11,D10,D9...D0
          CLR CS                   ; CS low to start conversion or keep low till done
          SETB DCLK                ; raise the clock
          MOV C,SDAT               ; put data into C flag
          RLC A                    ; shift C into Acc (A/D low bits)
          XCH A,ADRESH             ; get ADRESH byte(sav low bits in ADRESH for now)
          RLC A                    ; shift C into Acc (A/D high bits)
          XCH A,ADRESH             ; get low bits back into Acc for next loop
          DJNZ COUNTA,NXTBIT
          MOV ADRESL,A             ; put A into ADRESL
          ANL ADRESH,#0FH          ; mask off unwanted bits (x,X,X,Null
          SETB CS                  ; set CS hi to end conversion
;=END__GET_AD=========================================================
;=====================================================================


;=====================================================================
PROCDIGS:
          CALL BIN16BCD
          MOV  R0,#7
NXTDIG:
          MOV  A,#30H
          ADD  A,@R0
          CALL SENDCHAR
          DEC  R0
          CJNE R0,#3,NXTDIG

          CALL RETNEWLINE          ; send a carrage return and line feed
          CALL DELAY1              ; wait here awhile
          JMP START

;=====================================================================
;=SUBROUTINES=========================================================
;=====================================================================
;=====================================================================
;=====================================================================
RETNEWLINE:
          MOV A,#0AH               ; *** \n newline
          CALL SENDCHAR
          MOV A,#0DH               ; *** return
          CALL SENDCHAR
          RET
;=====================================================================
SENDCHAR:
T_TST:    JNB TI,T_TST             ; loop till output complete
          CLR TI                   ; clear bit
          MOV SBUF,A               ; send data
          RET
;=====================================================================
```

```
;*****************************************************************************
;  BIN16BCD

;   The following routine converts an unsigned integer value in the
;   range of 0 - 9999 to an unpacked Binary Coded Decimal number. No
;   range checking is performed.
;
;   INPUT: R3 (MSB), R2(LSB) contain the binary number to be
;       converted.
;  OUTPUT: R7(MSD), R6, R5, R4(LSD) contain the 4 digit, unpacked BCD
;       representation of the number.
; Uses: R1,R2,R3,R4,R5,R6,R7,ACC
;*****************************************************************************

BIN16BCD:

         MOV  R1,#16D            ; loop once for each bit (2 bytes worth)
         MOV  R5,#0              ; clear regs.
         MOV  R6,#0
         MOV  R7,#0

BCD_16LP:

         MOV  A,R2
         ADD  A,R2
         MOV  R2,A

         MOV  A,R3
         ADDC A,R3
         MOV  R3,A
;======
         MOV  A,R5
         ADDC A,R5
         DA   A
         MOV  R5,A

         MOV  A,R6
         ADDC A,R6
         DA   A
         MOV  R6,A
         DJNZ R1,BCD_16LP        ; loop until all 16 bits done
;=================
;unpack the digits
;=================
         SWAP A                  ;swap so that digit 4 is rightmost
         ANL  A,#0FH             ;mask off digit 3
         MOV  R7,A               ;save digit 4 in R7
         MOV  A,R6               ;get digits 3,4 again
         ANL  A,#0FH             ;mask off digit 4
         MOV  R6,A               ;save digit 3

         MOV  A,R5               ;get digits 1,2
         SWAP A                  ;swap so that digit 2 is rightmost
         ANL  A,#0FH             ;mask off digit 1
         XCH  A,R5               ;put digit 2 in R5, digit 1 => ACC
         ANL  A,#0FH             ;mask off digit 2
         MOV  R4,A               ;save digit 1 in R4 then exit

        RET
;=====================================================================

DELAY1:  DJNZ R2,DELAY1
DELAY2:  DJNZ R3,DELAY1
         RET
;=====================================================================
;=====================================================================
END
```

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX B: SYNCHRONOUS PORT SOURCE CODE

```
;
;
$MOD51
$TITLE(ads2)
$DATE(7/29/98)
$PAGEWIDTH(132)
$OBJECT(C:\ASM51\ads2.OBJ)
;
; Author: Lee Studley
; Assembled with Metalink's FreeWare ASM51 assembler
; Tested with NOICE emulation software.
; Tested with a DALLAS DS80C320 (8031) micro clocked @ 11.0592mhz
; This micro has a 2nd UART resource at pins P1.2,P1.3
;
; This test uses a the UART MODE0 approach yielding a conversion
; time of approximately 43.4uS
; The result is transmitted via the original 8051 UART to an ascii
; terminal at 19.2k baud 8N1 format
;
;================= RESET AND INTERRUPT VECTORS ======================
;
RSTVEC      EQU 0000H               ;
IE0VEC      EQU 0003H               ;
TF0VEC      EQU 000BH               ;
IE1VEC      EQU 0013H               ;
TF1VEC      EQU 001BH               ;
RITIVEC     EQU 0023H               ;
TF2VEC      EQU 002BH               ; ( 8052 )
;
;================= VARIABLES =================
            DSEG

;================= PROGRAM VARIABLES =================
COUNTA      EQU 30H
COUNTB      EQU 31H
ADRESL      EQU 2
ADRESH      EQU 3

;================= HARDWARE EQUATES =================
DCLK        EQU P1.3
SDAT        EQU P1.2
CS          EQU P1.1
;
;

;2nd Uart equates
SCON1       EQU 0C0H
SBUF1       EQU 0C1H
REN_1       BIT SCON1.4
R1_1        BIT SCON1.0
;
;
;================= CONSTANTS =================
;
;================= PROGRAM CODE =================
;
            CSEG

;           ORG RSTVEC
;           LJMP START
            ORG 4000H               ; NOICE SRAM/PROGRAM SPACE
```

```
;=====================================================================
START:
;=====================================================================
;  Initialize the on-chip serial port for mode 1
;  Set timer 1 for baud rate: auto reload timer
;=====================================================================
SETUPUART:
            MOV  PCON,#80H                ; SET FOR DOUBLE BAUD RATE
            MOV  TMOD,#00100010B          ; two 8-bit auto-reload counters
            MOV  TH1,#0FDH                ; 19.2K @ 11.059 MHZ
            MOV  SCON,#01010010B          ; mode 1, TI set
            SETB TR1                      ; start timer for serial port
;=====================================================================
SETUPUART2:
            MOV  SCON1,#00000000B    ; 2nd uart mode 0, TI set
                                     ; Shift clk(TX)=Tosc/12
;=====================================================================

;=====================================================================
; GET_AD: Initiates the A/D conversion and retreives the AD sample into
; ADRESH,ADRESL.
; The A/D convertor is connected to port1 pins 1..3 as:
; DCLK      EQU P1.3 Tx(synchronous clock)
; SDAT      EQU P1.2 Rx(synchronous data)
; CS        EQU P1.1 I/O
; Uses: ADRESL,ADRESH,ACC,COUNTA
; Exits: ADRESH=(B11..B4), ADRESL(B3..B0,x,x,x,x)
;=====================================================================
GET_AD:     SETB CS                   ; set cs hi
            CLR  DCLK                 ; X,X,NULL,D11,D10,D9...D0
            CLR  CS                   ; CS low to start conversion or keep low till done
            SETB DCLK                 ; 1st S/H clock
            CLR  DCLK                 ;
            SETB DCLK                 ; 2nd S/H clock and leave DCLK high

            SETB REN_1                ; REN=1 & R1_1=0 initiates a receive
            CLR  R1_1                 ;

BYTE_1:     JNB  R1_1,BYTE_1
            MOV  A,SBUF1              ; toss this byte
            CLR  R1_1
BYTE_2:     JNB  R1_1,BYTE_2
            MOV  ADRESL,SBUF1         ; save lsbs
            CLR  R1_1
BYTE_3:     JNB  R1_1,BYTE_3
            MOV  ADRESH,SBUF1         ; save msbs
            SETB CS                   ; set CS hi to end conversion
            ANL  ADRESL,#0F0H         ; mask off unwanted lsb bits

;=END__GET_AD=========================================================
;=====================================================================



;=====================================================================
PROCDIGS:
            CALL BIN16BCD
            MOV  R0,#7
NXTDIG:
            MOV  A,#30H
            ADD  A,@R0
            CALL SENDCHAR
            DEC  R0
            CJNE R0,#3,NXTDIG

            CALL RETNEWLINE           ; send a carrage return and line feed
```

```
            CALL DELAY1             ; wait here awhile
            JMP  START

;*****************************************************************
;=SUBROUTINES====================================================
;*****************************************************************
;*****************************************************************
;*****************************************************************
RETNEWLINE:
            MOV  A,#0AH             ; *** \n newline
            CALL SENDCHAR
            MOV  A,#0DH             ; *** return
            CALL SENDCHAR
            RET
;*****************************************************************
SENDCHAR:
T_TST:      JNB  TI,T_TST           ; loop till output complete
            CLR  TI                 ; clear bit
            MOV  SBUF,A             ; send data
            RET
;*****************************************************************

;*****************************************************************
;  BIN16BCD
;   The following routine converts an unsigned integer value in the
;   range of 0 - 9999 to an unpacked Binary Coded Decimal number. No
;   range checking is performed.
;
;   INPUT: R3 (MSB), R2(LSB) contain the binary number to be
;       converted.
;  OUTPUT: R7(MSD), R6, R5, R4(LSD) contain the 4 digit, unpacked BCD
;       representation of the number.
; Uses: R1,R2,R3,R4,R5,R6,R7,ACC
;*****************************************************************************

BIN16BCD:
            MOV  A,ADRESL           ; right justify the
            SWAP A                  ; R3:R2 pair for bin16bcd routine
            MOV  ADRESL,A

            MOV  A,ADRESH
            SWAP A
            ANL  A,#0F0H
            ORL  ADRESL,A

            MOV  A,ADRESH
            SWAP A
            ANL  A,#0FH
            MOV  ADRESH,A
;======
            MOV  R1,#16D            ; loop once for each bit (2 bytes worth)
            MOV  R5,#0              ; clear regs.
            MOV  R6,#0
            MOV  R7,#0

BCD_16LP:

            MOV  A,R2
            ADD  A,R2
            MOV  R2,A

            MOV  A,R3
            ADDC A,R3
            MOV  R3,A
;======
            MOV  A,R5
```

```
            ADDC A,R5
            DA   A
            MOV  R5,A

            MOV  A,R6
            ADDC A,R6
            DA   A
            MOV  R6,A
            DJNZ R1,BCD_16LP        ; loop until all 16 bits done
;=================
;unpack the digits
;=================
            SWAP A                  ;swap so that digit 4 is rightmost
            ANL  A,#0FH             ;mask off digit 3
            MOV  R7,A               ;save digit 4 in R7
            MOV  A,R6               ;get digits 3,4 again
            ANL  A,#0FH             ;mask off digit 4
            MOV  R6,A               ;save digit 3

            MOV  A,R5               ;get digits 1,2
            SWAP A                  ;swap so that digit 2 is rightmost
            ANL  A,#0FH             ;mask off digit 1
            XCH  A,R5               ;put digit 2 in R5, digit 1 => ACC
            ANL  A,#0FH             ;mask off digit 2
            MOV  R4,A               ;save digit 1 in R4 then exit

            RET

;=====================================================================
;=====================================================================
;=====================================================================

DELAY1:     DJNZ R2,DELAY1
DELAY2:     DJNZ R3,DELAY1
            RET
;=====================================================================
;=====================================================================
;=====================================================================

END
```

NOTES:

AN703

Using the MCP320X 12-Bit Serial A/D Converter with Microchip PICmicro® Devices

Author: Jake McKernan
Microchip Technology Inc.

OVERVIEW

The MCP320X devices comprise a family of 12-bit successive approximation Analog to Digital (A/D) Converters. These devices provide from one to eight analog inputs with both single ended and differential inputs. Data is transferred to and from the MCP320X through a simple SPI™-compatible 3-wire interface. This application note discusses how to interface the MCP320X devices to Microchip PICmicro® devices, using both software and hardware SPI with examples shown in C and Assembly languages. The programs in this application note were developed using a PIC16C62A and MCP3202 on a PICDEM-2 demonstration board. As a matter of convenience, the CLK, DO, and DI pins of the PIC16C62A are used for all examples, whether using the hardware SPI peripheral or the software SPI implementation. The software SPI may be adapted to I/O ports on any PICmicro device.

COMMUNICATION

Communication to the MCP3202 is accomplished via a synchronous SPI-compatible scheme. This interface consists of three lines; DOUT, DIN and CLK. Control information is loaded into the MCP320X through the DIN line and data is output on the DOUT line. The CLK signal is generated by the PICmicro and is used as both communication and conversion clock for the A/D Converter. Data bits are latched in from DIN on the rising edge of CLK and latched out to DOUT on the falling edge. A fourth line, $\overline{CS}$, is an active low signal used to select the chip and enable it for conversion and communication. See Figure 1 for a communication timing diagram.

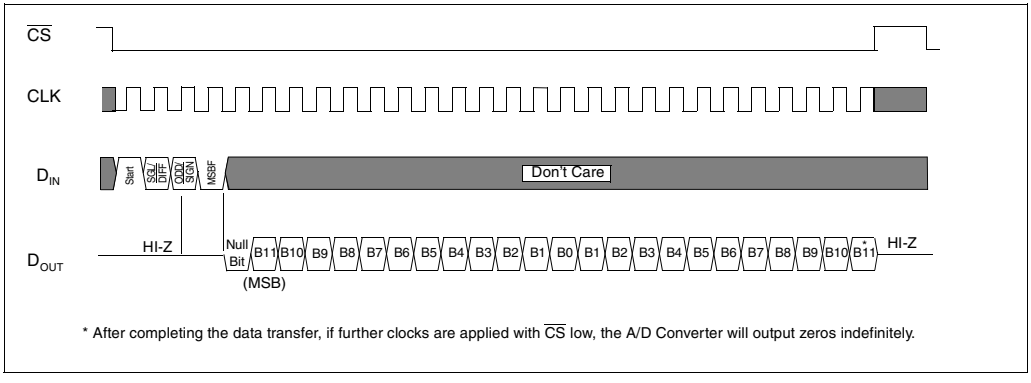


FIGURE 1: *Communication with MCP3202 using LSB first format*

SPI is a trademark of Motorola Corporation

A 4-bit configuration command is issued to the MCP3202 to begin the conversion process. When communication of the command word to the MCP3202 begins, the first '1' bit seen by the MCP3202 on the DIN line will be interpreted as a start bit. Leading 0's may be clocked into the device with no effect. The start bit is followed by a mode selection bit, indicating whether the conversion result will be single-ended or differential. A mode select bit of '1' selects single-ended mode and '0' selects differential mode. Next, the channel select bit is clocked into the MCP3202, which sets the channel to be converted. A '0' in this bit position selects Channel 0, while a '1' selects Channel 1. If differential mode was selected, the channel select bit determines which channel will be subtracted from the other. Table 1 illustrates how the A/D result will be affected by the channel and mode selection bits. Finally, a data format bit is clocked into the MCP3202. This bit selects whether the result of the conversion will be shifted out in LSb format. A '0' in this bit position will cause the data to be shifted out in MSb only format. If a '1', the data will first be shifted out in MSb format, followed by the same data in LSb format. Keep in mind that the data will *always* be shifted out in MSb format, regardless of the state of the data format bit.

	CONFIG BITS		**CHANNEL SELECTION**		**GND**
	SGL/$\overline{\text{DIFF}}$	ODD/$\overline{\text{SIGN}}$	0	1	
SINGLE ENDED MODE	1	0	+		-
	1	1		+	-
PSEUDO-DIFFERENTIAL MODE	0	0	IN+	IN-	
	0	1	IN-	IN+	

TABLE 1: *Configuration Bits for the MCP3202*

The command word is followed by the clocking in of a dummy bit, during which time the converter determines whether the MSb should be a 0 or 1. The 12-bit A/D result is then clocked out of the MCP3202 one bit at a time. The LSb of the A/D result is common to both data formats, i.e. the LSb is output only once while all other result bits are output twice (once for MSb first format, once for LSb first format). 0's will be clocked out of the DOUT line if CLK pulses are issued after all data bits are extracted from the converter.

IMPLEMENTATION

As previously mentioned, several code examples of interfacing to the MCP3202 are shown in this application note. All methods use essentially the same algorithm of performing an A/D conversion, displaying the result on PORTB, then waiting for a keypress. The examples cover hardware and software SPI, relocatable and absolute assembly and C.

Written in absolute assembly, Appendix A shows the use of the hardware SSP module in master SPI mode. The SSP is set up to clock data in on the rising edge, clock data out on the falling edge and drive the clock high when idle, with a frequency of Fosc/64. All bits of PORTB are configured as outputs and the port is cleared. To begin the conversion process, the MCP3202 is selected using the $\overline{\text{CS}}$ line and 0x01 is loaded into the SSPBUF of the PIC16C62A. This shifts out seven leading 0's, followed by a start bit. The subroutine WAIT_BF then monitors the BF flag in the SSPSTAT register, which indicates when the 8-bit transfer is complete. Next, a value of 0xE0 is loaded into the SSPBUF, the MSb's being the three configuration information bits, and the lower five bits being dummy information to round out the byte. The configuration bits in this example set the MCP3202 up for single-ended conversion on channel 1, with the output in MSb first format. During the transmission of the 5 LSb's, the MCP3202 will begin shifting out A/D result data. The WAIT_BF subroutine is called after the SSPBUF is loaded, waiting for the transmission to be complete. Once the transmission is complete, the MSb's of the result are read from the SSPBUF, masked, and displayed on PORTB for examination by the user. Finally, a dummy value of 0x00 is loaded into the SSPBUF to retrieve the final eight LSb's of the A/D result from the MCP3202.

The WAIT_PRESS routine is then called, waiting for the RA4 button of the PICDEM-2 board to be pressed and released. Once the button has been pressed and released, the remaining data is read from the SSPBUF and displayed on the PORTB pins. This information is displayed until the RA4 button is again pressed and released (by calling the WAIT_PRESS subroutine), after which the A/D process begins again.

Appendix B demonstrates the same functionality as the program in Appendix A, but is written in the C language. This allows portability between platforms (12-bit, 14-bit or 16-bit cores), with a minimum of change to the program.

Appendices C and D are used together to show a hardware SPI implementation using relocatable assembly code. The main file (MCP3202c.asm) is shown in Appendix C and contains the main functionality of the program, while the assembly file shown in Appendix D (waitfcn.asm) contains the auxiliary functions (i.e. waiting for SPI transmission to complete and for RA4 press and release). The linker script (16c62a.lkr) shown in Appendix D controls where the relocatable segments

are placed in the PIC16C62A program memory and defines the processor's available RAM space for the linker. Please consult the MPASM User's Guide for more details on how to write relocatable code.

Appendix E illustrates communication to the MCP3202 using firmware SPI rather than the hardware peripheral. The same I/O pins are used to generate the clock and data signals as with the hardware peripheral, for convenience. Program initialization occurs as with the previous examples, except that the hardware peripheral is excluded and replaced with initialization of PORTC bits. Three registers are initialized to be used as input and output buffers, and there are two new subroutines added to communicate to the MCP3202. The first routine called will be OUT_CONTROL, which issues the control word to the MCP3202. The control word to be sent is loaded into the OUTBUF register before the subroutine is called. Each of the four bits is then shifted out and clocked into the A/D Converter using the DOUT and CLK lines of PORTC, respectively. Once all bits are shifted out, the subroutine returns to the calling function. To retrieve the data from the A/D Converter, a second subroutine is implemented. The IN_DATA subroutine toggles the CLK line and reads the DIN line, shifting each new bit into the INBUFL and INBUFH registers. All 12 bits of the result are read by this subroutine which will return to the calling function once the transfer is complete. As with the previous examples, the MSb's are displayed on PORTB, while the program waits for RA4 to toggle. The LSb's are then displayed, the program waits for RA4 to toggle again, and the process repeats again.

Appendix F is a variation on Appendix E, demonstrating the use of relocatable assembly to implement a software SPI. The same subroutines are used for this example, but are declared as external. The wait functions and linker script (waitfcn.asm, 16c62a.lkr) files shown in Appendix C are used in this example. The ser_io.asm file shown in Appendix G contains the OUT_CONTROL and IN_DATA subroutines used in this example.

The final example, shown in Appendix H, illustrates the firmware SPI implementation in the C language. Two functions are added to this implementation, Output_Control and Input_Data. As with the previous example, the Output_Control shifts the 4-bit command out to the MCP3202 one bit at a time and Input_Data reads all 12 bits of the result. The data is then displayed on PORTB, waiting for input on RA4 before continuing on. In this program, the A/D result data may be accessed in one of two ways; as a 16-bit value or as two 8-bit values. When reading the value in from the MCP3202 using the Input_Data function, the A/D result is treated as a 16-bit value. During the display portion of the program, the result is accessed 8-bits at a time for display on PORTB.

SCHEMATIC

The code for this application note was developed on a PICDEM-2 demonstration board. An equivalent circuit of the board as used in this application note is shown in Appendix I. A full schematic of the PICDEM-2 board can be found in the PICDEM-2 User's Guide, available with the kit or from the Microchip web site (www.microchip.com).

The SPI communication lines CLK, DOUT and DIN are connected to RC3, RC4 and RC5, respectively. The $\overline{CS}$ signal is generated using RC2 as a general purpose output pin. PORTB is used entirely as an output port for display of A/D result data. All LED's are driven through 470Ω current limiting resistors. RA4 is connected to a momentary contact switch and pullup resistor for allowing the user to cycle through the A/D result data on PORTB.

Channel 1 of the A/D Converter is used throughout the application note, and must have an analog voltage applied to it to get meaningful results from the MCP3202. This was done using a 0-5v power supply output fed directly into pin three of the MCP3202.

The PIC16C62A uses the RC oscillator configuration as the main clock, operating at an approximate frequency of 4MHz. An RC network is also provided on the MCLR line to help ensure that the device is reset correctly on application of power.

CONCLUSION

The example code shown in this application note gives a firm grasp of how to interface the MCP3202 A/D Converter to PICmicro devices. The code has the potential to be adapted to any Microchip PICmicro device, an exercise left up to the user. Implementations in multiple languages and styles also gives the developer flexibility in successfully writing code and libraries to use this device in end-user applications.

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX A: HARDWARE SPI, ABSOLUTE ASSEMBLY

```
;*******************************************************************************
;*
;*     This program demonstrates communication with the MCP3202 A/D converter
;*     using absolute assembly code. This code was written for the midrange
;*     PICmicro devices (using a PICDEM-2 board and the 16C62A) and uses the SSP
;*     module in SPI mode for communication to the MCP3202.
;*
;*     Filename: mcp3202a.asm
;*
;*     (C) 1998 Microchip Technology, Inc.
;*     All Rights Reserved
;*
;*******************************************************************************

   list p=16c62a

   include "p16c62a.inc"


ADCS   equ    0x02                ;chip select line for A/D


   ORG 0x0000

   clrf PCLATH                ;reset PCLATH for Page0 operation
   clrf STATUS                ;reset STATUS for Bank 0 operation
   clrf FSR                   ;clear FSR
   goto START                 ;begin main program

   ORG 0x0004
_ISR
   goto _ISR                  ;stay here if interrupt occurs

WAIT_BF
      bsf STATUS,RP0             ;select Bank0
      btfss SSPSTAT,BF           ;check for BF set
      goto WAIT_BF               ;continue to wait
      bcf STATUS,RP0             ;select Bank1
      return                     ;return to caller


WAIT_PRESS
      btfsc PORTA,4              ;check for button press
      goto WAIT_PRESS

WAIT_RLS
      btfss PORTA,4              ;check for button release
      goto WAIT_RLS
      return                     ;return to caller


START
      movlw 0x32                 ;set up SSP to clock data out on falling edge
      movwf SSPCON               ;clock data in on rising edge, clock idle high

      clrf PORTB                 ;clear PortB outputs
```

```
   bsf STATUS,RP0              ;select Bank1
   movlw 0x10
   movwf TRISC                 ;set up Port C for SPI master

   clrf TRISB                  ;configure PortB as outputs

   bcf STATUS,RP0              ;select Bank0
   bsf PORTC,ADCS              ;deselect A/D device

BEGIN_AD
      bcf PORTC,ADCS              ;select A/D device
      movlw 0x01
      movwf SSPBUF                ;output start bit

      call WAIT_BF                ;wait for transfer complete

      movlw 0xE0                  ;output 3 command and 5 dummy bits
      movwf SSPBUF                ;shift out command and receive 4 MSb's
      call WAIT_BF                ;wait for transfer complete

      movf SSPBUF,W               ;read result (MSB's of conversion)
      andlw 0x0F                  ;mask out MSb's
      movwf PORTB                 ;display on PortB

      movlw 0x00                  ;load dummy value
      movwf SSPBUF                ;shift remaining bits
      call WAIT_BF                ;wait for transfer complete

      call WAIT_PRESS             ;wait for button press/release before advancing

      movf SSPBUF,W               ;read result (LSb's)
      movwf PORTB                 ;display on PortB

      bsf PORTC,ADCS              ;de-select A/D converter

      call WAIT_PRESS             ;wait for button press/release before advancing

HERE
      goto BEGIN_AD               ;play it again, Sam


      END
```

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX B: HARDWARE SPI, C LANGUAGE

```
/************************************************************************
*
*  This program is written to demonstrate interfacing the MCP3202 A/D
*  converter to Microchip PICmicro devices.  The code demonstrates
*  how to implement hardware SPI to communicate with the converter,
*  and is written in C for the HiTech PICC C compiler.  By modifying the
*  #include statement to "#include<16c62a.h>" the code may be compiled
*  using MPLAB-C 1.21.
*
*  Filename: mcp3202b.c
*
*  (C) 1998 Microchip Technology, Inc.
*  All Rights Reserved
*
************************************************************************/


#include<pic1662.h>       /* modify this statement for use with the MPLAB-C compiler */


#define ADCS 0x04        /* I/O bit position for CS line */
#define BUSY 0x01        /* Bit0 of SSPSTAT, indicated when SPI xmission complete */
#define BUTTON 0x10      /* I/0 bit position for RA4 line */


void Wait_for_Press()
{
   while(PORTA & BUTTON)
   {
      /* wait for button press */
   }

   while(!(PORTA & BUTTON))
   {
      /* wait for button release */
   }
}


void main(void)
{
   TRISB = 0x00;
   PORTB = 0x00;         /* reset PortB outputs */

   SSPCON = 0x32;        /* set up SSP to clock data out on falling edge */
   TRISC = 0x10;         /* clock data in on rising edge, clock idle high */

   PORTC |= ADCS;        /* de-select A/D device */

   while(1)
   {
      PORTC &= ~ADCS;    /* select A/D device */

      SSPBUF = 0x01;     /* output start bit */

      while(!(SSPSTAT & BUSY))
```

```
      {
                        /* wait for transfer complete */
      }

      SSPBUF = 0xE0;   /* output 3 command, 5 dummy bits */

      while(!(SSPSTAT & BUSY))
      {
                       /* wait for transfer complete */
      }

      PORTB = SSPBUF & 0x0F;      /* mask and output conversion MSb's */

      SSPBUF = 0x00;       /* output dummy word */

      while(!(SSPSTAT & BUSY))
      {
                           /* wait for transfer complete */
      }

      PORTC |= ADCS;       /* de-select A/D device */

      Wait_for_Press();    /* wait for button press/release */

      PORTB = SSPBUF;      /* output LSb's */

      Wait_for_Press();    /* wait for button press/release */
   }
}
```

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX C: HARDWARE SPI, RELOCATABLE ASSEMBLY

```
;*******************************************************************************
;*
;*     This program demonstrates communication with the MCP3202 A/D converter
;*     using relocatable assembly code.  This code was written for the midrange
;*     PICmicro devices (using a PICDEM-2 board and the 16C62A) and uses the SSP
;*     module in SPI mode for communication to the MCP3202.
;*
;*     The two subroutines WAIT_BF and WAIT_PRESS are external functions, compiled
;*     and linked separately from the WAITFCN.ASM file.  These subroutines wait
;*     for the SPI transmission to complete and for RA4 to be pushed and released,
;*     respectively.
;*
;*     Filename: mcp3202c.asm
;*
;*     (C) 1998 Microchip Technology, Inc.
;*     All Rights Reserved
;*
;*******************************************************************************

list p=16C62a

#include "p16c62a.inc"

ADCSequ0x02                         ;CS line for MCP3202 (RC6)

EXTERN WAIT_BF                      ;define wait function call symbols
EXTERN WAIT_PRESS


RESETCODE                           ;select reset code section

clrf PCLATH                         ;reset PCLATH on powerup
clrf STATUS                         ;reset STATUS on powerup
clrf FSR                            ;reset FSR on powerup
goto START                          ;go start and initialize program

INTCODE                             ;select interrupt code section
_ISR
goto _ISR                           ;stay here if interrupt occurs


START                               ;initialization
movlw 0x32
movwf SSPCON                        ;setup SSP for operation

clrf PORTB                          ;reset LED output port

bsf STATUS,RP0                      ;select Bank1
movlw 0x10
movwf TRISC                         ;configure PORTC for operation

clrf TRISB                          ;configure PORTB as outputs

bcf STATUS,RP0                      ;select Bank0
bsf PORTC,ADCS                      ;deselect A/D converter
```

```
BEGIN_AD                        ;start A/D conversion
bcf PORTC,ADCS                  ;select A/D converter
movlw 0x01                      ;load start bit
movwf SSPBUF                    ;output start bit to A/D

call WAIT_BF                    ;wait for transmission complete

movlw 0xE0                      ;load 3 command and 5 dummy bits
movwf SSPBUF                    ;output on SPI port

call WAIT_BF                    ;wait for transmission complete

movf SSPBUF,W                   ;read A/D result MSb's
   andlw 0x0F                   ;mask off garbage bits
movwf PORTB                     ;output MSb's on PORTB LED's

movlw 0x00                      ;load dummy data
movwf SSPBUF                    ;output on SPI (shifts in LSb's)
call WAIT_BF                    ;wait for transmission complete

call WAIT_PRESS                 ;wait for button press/release

movf SSPBUF,W                   ;read A/D result LSb's
movwf PORTB                     ;output LSb's on PORTB LED's

bsf PORTC,ADCS                  ;deselect A/D converter

call WAIT_PRESS                 ;wait for button press/release

HERE
goto BEGIN_AD                   ;repeat process

END
```

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX D: WAIT FUNCTIONS AND LINKER SCRIPT FOR APPENDIX C

```
;*****************************************************************************
;*
;*     Wait functions for MCP3202 A/D converter demonstration.  These
;*     functions wait for SPI communication and RA4 button press/release
;*     on the PICDEM-2 board.  This file is to be assembled and linked
;*     with mcp3202c.ASM or mcp3202e.ASM for proper usage.
;*
;*     Filename: waitfcn.asm
;*
;*     (C) 1998 Microchip Technology, Inc.
;*     All Rights Reserved
;*
;*****************************************************************************
       list p=16C62a
       #include "p16c62a.inc"
       CODE                            ;select code section

WAIT_BF                                ;wait for SPI transmission complete
       GLOBAL WAIT_BF                  ;declare WAIT_BF visible to outside world
       bsf STATUS,RP0                  ;select Bank1
       btfss SSPSTAT,BF                ;check for transmission complete (BF set)
       goto WAIT_BF                    ;not finished, continue waiting
       bcf STATUS,RP0                  ;select Bank0
       return                          ;return to calling function

WAIT_PRESS                             ;wait for RA4 press/release
       GLOBAL WAIT_PRESS                 ;declare WAIT_PRESS visible to outside world

       btfsc PORTA,4                   ;check for button press
       goto WAIT_PRESS                 ;not pressed, check again

WAIT_RLS                               ;button now pressed
       btfss PORTA,4                   ;check for button release
       goto WAIT_RLS                   ;not released, check again
       return                           ;button now released, return to calling func

       END

//*****************************************************************************
//*
//*      16C62A Linker Script to be used with MCP3202C.ASM and WAITFCN.ASM
//*      to link the corresponding object files.
//*
//*      Filename: 16c62a.lkr
//*
//*      (C) 1998 Microchip Technology, Inc.
//*      All Right Reserved
//*
//*****************************************************************************

CODEPAGE NAME=reset_vector START=0x00 END=0x03
CODEPAGE NAME=interrupt_vector START=0x04 END=0x7FF
DATABANK   NAME=gpr0     START=0x20     END=0x7F
DATABANK   NAME=gpr1     START=0xA0     END=0xBF
DATABANK   NAME=sfr0     START=0x0      END=0x1F     PROTECTED
DATABANK   NAME=sfr1     START=0x80     END=0x9F     PROTECTED
SECTION NAME=RESET ROM=reset_vector
SECTION NAME=INT ROM=interrupt_vector
```

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX E: FIRMWARE SPI, ABSOLUTE ASSEMBLY

```
;*******************************************************************************
;*
;*     This program demonstrates communication with the MCP3202 A/D converter
;*     using absolute assembly code.  This code was written for the midrange
;*     PICmicro devices (using a PICDEM-2 board and the 16C62A) and uses firmware
;*     to implement the SPI module for communication to the MCP3202.
;*
;*     Filename: mcp3202d.asm
;*
;*     (C) 1998 Microchip Technology, Inc.
;*     All Rights Reserved
;*
;*******************************************************************************
list p=16c62a

include "p16c62a.inc"

ADCS    equ     0x02                ;chip select line for A/D converter
DOUT    equ     0x05                ;serial data out to A/D converter
DIN     equ     0x04                ;serial data in from A/D converter
CLK     equ     0x03                ;serial data clock to A/D converter

        CBLOCK 0x20
OUTBUF
INBUFH
INBUFL
COUNT
        ENDC

ORG 0x0000

clrf PCLATH                 ;reset PCLATH for Page0 operation
clrf STATUS                 ;reset STATUS for Bank 0 operation
clrf FSR                    ;clear FSR
goto START                  ;begin main program

ORG 0x0004
_ISR
goto _ISR                   ;stay here if interrupt occurs

OUT_CONTROL
        movwf OUTBUF        ;load control word into buffer
        swapf OUTBUF        ;rotate control word into position

        movlw 0x04
        movwf COUNT         ;init bit counter

BIT_OUT
        rlf OUTBUF          ;rotate bit into carry
        bcf PORTC,DOUT      ;pre-clear data out
        btfsc STATUS,C      ;check if bit should be set
        bsf PORTC,DOUT      ;set data out

        bsf PORTC,CLK       ;generate clock pulse
        nop
        bcf PORTC,CLK
```

```
        decfsz COUNT          ;decrement bit counter
        goto BIT_OUT          ;output next bit
        return                   ;finished, return to caller


IN_DATA
        clrf INBUFH
        clrf INBUFL              ;reset input buffer

        movlw 0x0D
        movwf COUNT              ;init bit counter

BIT_IN
        bsf PORTC,CLK            ;set clock to latch bit
        bcf STATUS,C             ;pre-clear carry
        btfsc PORTC,DIN          ;check for high or low bit
        bsf STATUS,C             ;set carry bit

        rlf INBUFL
        rlf INBUFH               ;rotate bit into position

        bcf PORTC,CLK            ;drop clock for next bit

        decfsz COUNT             ;decrement bit counter
        goto BIT_IN              ;get next bit
        return                   ;return to caller


WAIT_PRESS
        btfsc PORTA,0x04         ;check for button press
        goto WAIT_PRESS

WAIT_RLS
        btfss PORTA,0x04         ;check for button release
        goto WAIT_RLS
        return                   ;return to caller


START
        clrf PORTB               ;clear PortB outputs

        movlw 0x40
        movwf PORTC              ;initialize PortC: ADCS high, DO, CLK low

bsf STATUS,RP0                   ;select Bank1
movlw 0x10
movwf TRISC                      ;set up Port C for SPI master

clrf TRISB                       ;configure PortB as outputs

bcf STATUS,RP0                   ;select Bank0

clrf OUTBUF                      ;reset output buffer
clrf INBUFH                      ;reset input buffer
clrf INBUFL

BEGIN_AD
        bcf PORTC,ADCS           ;select A/D converter
```

```
movlw 0x0F              ;load control word
call OUT_CONTROL        ;output control word

call IN_DATA                ;read data from A/D converter

bsf PORTC,ADCS              ;de-select A/D converter

movlw 0x0F                  ;load MSB mask
andwf INBUFH,W              ;mask out MSB's and put result in W
movwf PORTB                 ;output MSB's

call WAIT_PRESS             ;wait for button press

movf INBUFL,W               ;load LSB's into W
movwf PORTB                 ;output LSB's

call WAIT_PRESS             ;wait for button press
goto BEGIN_AD               ;play it again, Sam


END
```

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX F: FIRMWARE SPI, RELOCATABLE ASSEMBLY

```
;*******************************************************************************
;*
;*     This program demonstrates communication with the MCP3202 A/D converter
;*     using relocatable assembly code.  This code was written for the midrange
;*     PICmicro devices (using a PICDEM-2 board and the 16C62A) and uses the SSP
;*     module in SPI mode for communication to the MCP3202.
;*
;*     The subroutine WAIT_PRESS is an external function, compiled and linked
;*     separately from the WAITFCN.ASM file.  This subroutine waits for RA4 to be
;*     pushed and released.
;*     The subroutines OUT_CONTROL and IN_DATA are also external functions, but
;*     compiled and linked from the SER_IO.ASM file.  INBUFH and INBUFL are data
;*     bytes that are used by the IN_DATA routine to return the A/D conversion
;*     result to the calling function.
;*
;*     Filename: mcp3202e.asm
;*
;*     (C) 1998 Microchip Technology, Inc.
;*     All Rights Reserved
;*
;*******************************************************************************
list p=16c62a

include "p16c62a.inc"

       EXTERN WAIT_PRESS
       EXTERN OUT_CONTROL
       EXTERN IN_DATA

       EXTERN INBUFH
       EXTERN INBUFL

ADCS   equ    0x02          ;chip select line for A/D converter


RESET  CODE
clrf PCLATH                 ;reset PCLATH for Page0 operation
clrf STATUS                 ;reset STATUS for Bank 0 operation
clrf FSR                    ;clear FSR
goto START                  ;begin main program


INT    CODE
_ISR
goto _ISR                   ;stay here if interrupt occurs

START
       clrf PORTB           ;clear PortB outputs

       movlw 0x40
       movwf PORTC          ;initialize PortC: ADCS high, DO, CLK low

bsf STATUS,RP0              ;select Bank1
movlw 0x10
movwf TRISC                 ;set up Port C for SPI master
```

```
clrf TRISB                  ;configure PortB as outputs

bcf STATUS,RP0              ;select Bank0

BEGIN_AD
       bcf PORTC,ADCS               ;select A/D converter

       movlw 0x0F                   ;load control word
       call OUT_CONTROL             ;output control word

       call IN_DATA                 ;read data from A/D converter

       bsf PORTC,ADCS               ;de-select A/D converter

       movlw 0x0F                   ;load MSB mask
       andwf INBUFH,W               ;mask out MSB's and put result in W
       movwf PORTB                  ;output MSB's

       call WAIT_PRESS              ;wait for button press

       movf INBUFL,W                ;load LSB's into W
       movwf PORTB                  ;output LSB's

       call WAIT_PRESS              ;wait for button press
       goto BEGIN_AD                ;play it again, Sam


       END
```

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX G: RELOCATABLE ASSEMBLY FIRMWARE SPI FUNCTIONS FOR APPENDIX F

```
;*****************************************************************************
;*
;*     Serial functions for MCP3202 A/D converter demonstration.  These
;*     functions perform SPI communication.  This file is to be assembled
;*     and linked with mcp3202e.ASM for proper usage.
;*
;*     Filename: ser_io.asm
;*
;*     (C) 1998 Microchip Technology, Inc.
;*     All Rights Reserved
;*
;*****************************************************************************
list p=16c62a

#include "p16c62a.inc"


DOUT    equ     0x05                ;serial data out to A/D converter
DIN     equ     0x04                ;serial data in from A/D converter
CLK     equ     0x03                ;serial data clock to A/D converter

UDATA 0x20
OUTBUF res 1
INBUFH res 1
INBUFL res 1
COUNT  res 1

        GLOBAL INBUFH
        GLOBAL INBUFL

CODE

OUT_CONTROL
GLOBAL OUT_CONTROL
        movwf OUTBUF                ;load control word into buffer
        rlf OUTBUF
        rlf OUTBUF
        rlf OUTBUF
        rlf OUTBUF                  ;rotate control word into position

        movlw 0x04
        movwf COUNT                 ;init bit counter

BIT_OUT
        rlf OUTBUF                  ;rotate bit into carry
        bcf PORTC,DOUT              ;pre-clear data out
        btfsc STATUS,C              ;check if bit should be set
        bsf PORTC,DOUT              ;set data out

        bsf PORTC,CLK               ;generate clock pulse
        nop
        bcf PORTC,CLK

        decfsz COUNT                ;decrement bit counter
        goto BIT_OUT                ;output next bit
```

```
        return                      ;finished, return to caller


IN_DATA
GLOBAL IN_DATA
        clrf INBUFH
        clrf INBUFL                 ;reset input buffer

        movlw 0x0D
        movwf COUNT                 ;init bit counter

BIT_IN
        bsf PORTC,CLK               ;set clock to latch bit
        bcf STATUS,C                ;pre-clear carry
        btfsc PORTC,DIN             ;check for high or low bit
        bsf STATUS,C                ;set carry bit

        rlf INBUFL
        rlf INBUFH                  ;rotate bit into position

        bcf PORTC,CLK               ;drop clock for next bit

        decfsz COUNT                ;decrement bit counter
        goto BIT_IN                 ;get next bit
        return                      ;return to caller

        END
```

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX H: FIRMWARE SPI, C LANGUAGE

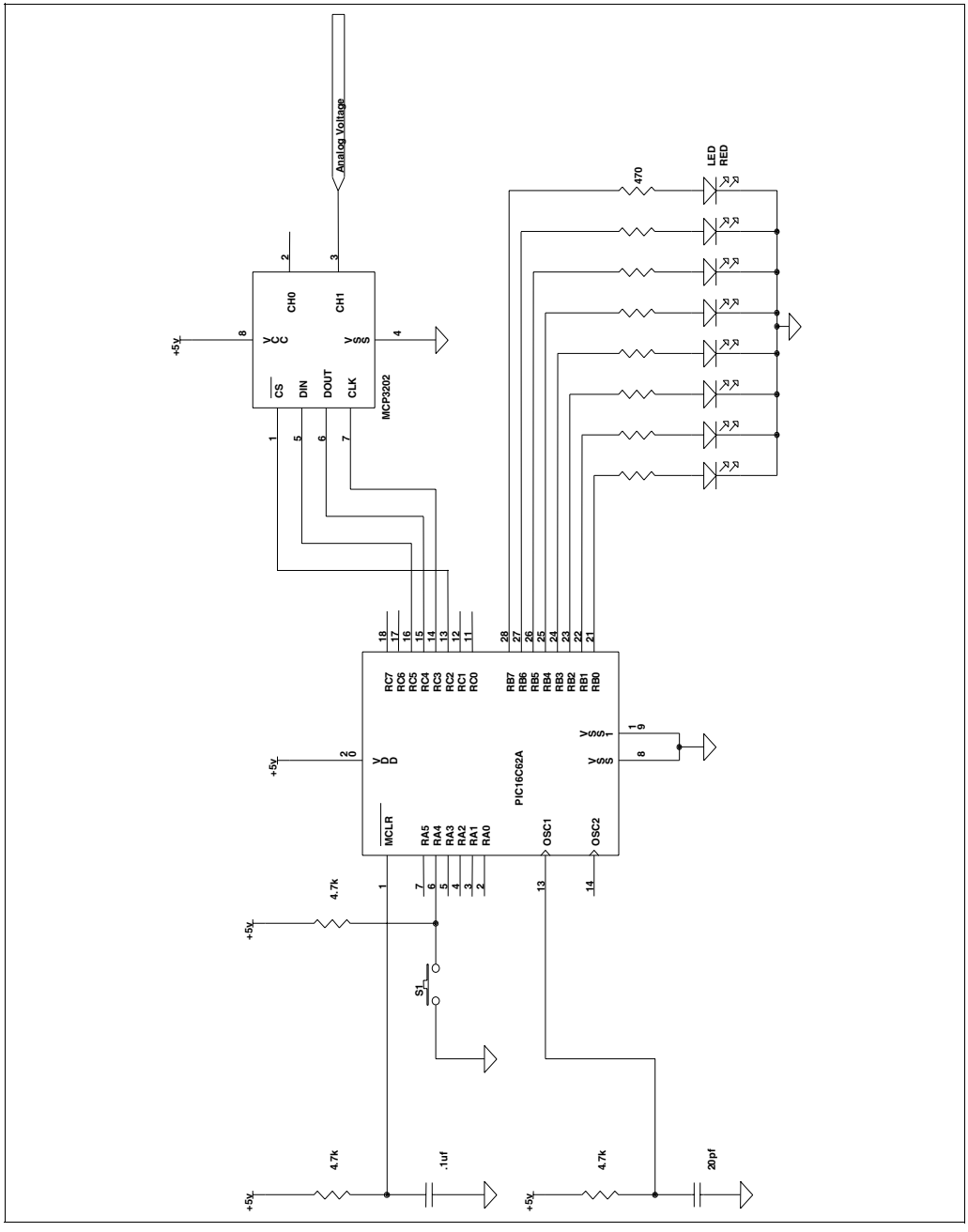
```
/************************************************************************
*
*  This program is written to demonstrate interfacing the MCP3202 A/D
*  converter to Microchip PICmicro devices.  The code demonstrates
*  how to implement software SPI to communicate with the converter,
*  and is written in C for the HiTech C compiler, PICC.  Changing the
*  #include directive to "#include<16c62a.h>" will allow the use of the
*  MPLAB-C v1.21 C compiler to compile this file.
*
*  Filename: mcp3202f.c
*
*  (C) 1998 Microchip Technology, Inc.
*  All Rights Reserved
*
************************************************************************/


#include <pic1662.h>    /* modify this statement for use with the MPLAB-C compiler */

#define ADCS 0x04       /* I/O bit position for CS line */
#define BUSY 0x01       /* Bit0 of SSPSTAT, indicated when SPI xmission complete */
#define BUTTON 0x10     /* I/0 bit position for RA4 line */

#define DOUT 0x20       /* data out to MCP3202 */
#define DIN 0x10        /* data in from MCP3202 */
#define CLK 0x08        /* clock out to MCP3202 */


/* Function Prototypes */

void Wait_for_Press(void);
void Output_Control(char TempChar);
int Input_Data(void);



void Wait_for_Press()
{
   while(PORTA & BUTTON)
   {
      /* wait for button press */
   }

   while(!(PORTA & BUTTON))
   {
      /* wait for button release */
   }
}


void Output_Control(char TempChar)
{
   unsigned char Mask = 0x08;       /* mask to test for 0/1 */
   unsigned char Count;             /* gen purpose bit counter */

   for(Count = 0x00; Count < 0x04; Count++)  /* count 4 bits */
```

```
   {
      PORTC &= ~DOUT;                /* pre-clear data line */

      if(TempChar & Mask)            /* check if bit should be high or low */
      {
         PORTC |= DOUT;              /* set data line */
      }

      PORTC |= CLK;                  /* send clock line high */

      Mask >>= 0x01;                 /* rotate mask for next bit */
                                     /* also used to burn time for clock */
      PORTC &= ~CLK;                 /* send clock line low */
   }
}



int Input_Data(void)
{
   unsigned char Count;              /* gen purpose bit counter */
   unsigned int Mask = 0x8000;       /* mask to insert '1' at bit position */
   unsigned int Result = 0x0000;     /* A/D result register */

   for(Count = 0x00; Count < 0x0D; Count++)  /* count 13 bits */
   {                                 /* 12-bit result + 1 null bit */
      if(PORTC & DIN)                /* check if bit is high or low */
      {
         Result |= Mask;             /* bit high, set bit in result */
      }

      PORTC |= CLK;                  /* send clock line high */

      Mask >>= 0x01;                 /* rotate mask for next bit */
                                     /* also used to burn time for clock */
      PORTC &= ~CLK;                 /* send clock line low */
   }

   Result >>= 0x03;                  /* rotate bits into position */
   Result &= 0x0FFF;                 /* mask out 12-bit result */

   return(Result);                   /* return result to caller */
}



void main(void)
{
   union DualAccess                     /* declare union to allow access to */
   {                                    /* variable as 8 or 16-bit */
      unsigned int By_16;               /* allows 16-bit access */

      struct Bytewise                   /* struct provides for 8-bit access */
      {
         unsigned char Lo;              /* LSB of variable */
         unsigned char Hi;              /* MSB of variable */
      } By_8;
   } ADresult;
```

```
TRISB = 0x00;
PORTB = 0x00;                          /* reset PortB outputs */

PORTC = 0x40;                          /* init PortC (A/D de-selected) */
TRISC = 0x10;                          /* config PortC */

PORTC |= ADCS;                         /* de-select A/D converter */

while(1)
{
   PORTC &= ~ADCS;                     /* select A/D converter */

   Output_Control((char)0x0F);         /* output control word to A/D converter */

   ADresult.By_16 = Input_Data();      /* read result from converter */

   PORTC |= ADCS;                      /* de-select A/D converter */

   PORTB = ADresult.By_8.Hi;           /* display A/D MSb's */

   Wait_for_Press();                   /* wait for key press/release */

   PORTB = ADresult.By_8.Lo;           /* display A/D LSb's */

   Wait_for_Press();                   /* wait for key press/release */

}
}
```

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX I: EQUIVALENT SCHEMATIC



NOTES:

AN704

Interfacing Microchip's MCP3201 Analog/Digital (A/D) Converter to MC68HC11E9-Based Microcontroller

Author: *Richard L. Fischer*
Microchip Technology Inc.

INTRODUCTION

Many of the embedded control systems designed today require some flavor of Analog-to-Digital (A/D) Converter. Embedded system applications such as Data Acquisition, Sensor Monitoring, and Instrumentation and Control all have varying A/D Converter requirements.

For the most part, these A/D Converter requirements are a combination of performance, cost, package size basis and availability. In some applications a microcontroller with an on-chip A/D Converter may meet the design requirements. Typically, these on-chip A/D Converter modules fit well into embedded applications which require a 10-35ksps A/D Converter. In other applications, a stand-alone A/D Converter is required for various performance reasons.

For those applications which require higher performance or remote sense capability, the Microchip MCP3201 12-bit A/D Converter fits very nicely.

The Microchip Technology Inc. MCP3201 employs a classic SAR architecture. The device uses an internal sample and hold capacitor to store the analog input while the conversion is taking place. Conversion rates of 100ksps are possible on the MCP3201. Minimum clock speed (10 kHz or 625sps, assuming 16 clocks) is a function of the capacitors used for the sample and hold.

The MCP3201 has a single pseudo-differential input. The (IN–) input is limited to ±100mV. This can be used to cancel small noise signals present on both the (IN+) and (IN–) inputs. This provides a means of rejecting noise when the (IN–) input is used to sense a remote signal ground. The (IN+) input can range from the (IN-) input to V_{REF}.

The reference voltage for the MCP3201 is applied to V_{REF} pin. V_{REF} determines the analog input voltage range and the LSB size, i.e.:

$$LSB\ size = \frac{V_{REF}}{2^{12}}$$

As the reference input is reduced, the LSB size is reduced accordingly.

Communication with the MCP3201 is accomplished using a standard SPI™ compatible serial interface. This interface allows direct connection to the serial ports of microcontrollers and digital signal processors.

The MCP3201 is suitable for use with a wide variety of microcontrollers from Microchip and others. This application note describes how to interface the MCP3201 with a Motorola MC68HC11 microcontroller. Application Note, AN702 covers microcontrollers based on the Intel 8051 architecture.

Figure 1 shows the hardware schematic for this interface. Appendix A contains a listing of the source code.

CIRCUIT DESCRIPTION

The serial interface of the Microchip MCP3201 A/D Converter has three wires, a serial clock input (DCLK), the serial data output (D_{OUT}) and the chip select input signal ($\overline{CS}$/SHDN). For this simple circuit interface, the Motorola MC68HC11E9 SPI port is used. PORTD:<4> is configured for the serial clock and PORTD:<2> is the data input to the microcontroller. The SPI clock rate for this application is set at 1 MHz.

The MC68HC11 is configured in the master mode with its CPOL and CPHA bits set to logic one (default setting on power-up).

A conversion is initiated with the high to low transition of $\overline{CS}$/SHDN (active low). The chip select is generated by PORTD:<5> of the microcontroller. The device will sample the analog input from the rising edge of the first clock after $\overline{CS}$ goes low for 1.5 clock cycles. On the falling edge of the second clock, the device will output a low null bit. With the next 12 clocks, the MCP3201 will output the result of the conversion with the MSB first (See Figure 2 and Figure 3). Data is always output from the device on the falling edge of the clock. If the device continues to receive clocks while $\overline{CS}$/SHDN is low, the device will output the conversion LSB first. If more clocks are provided to the device while $\overline{CS}$/SHDN is still low (after the LSB first data has been transmitted), the device will clock out zeros indefinitely.

SPI is a trademark of Motorola Inc.

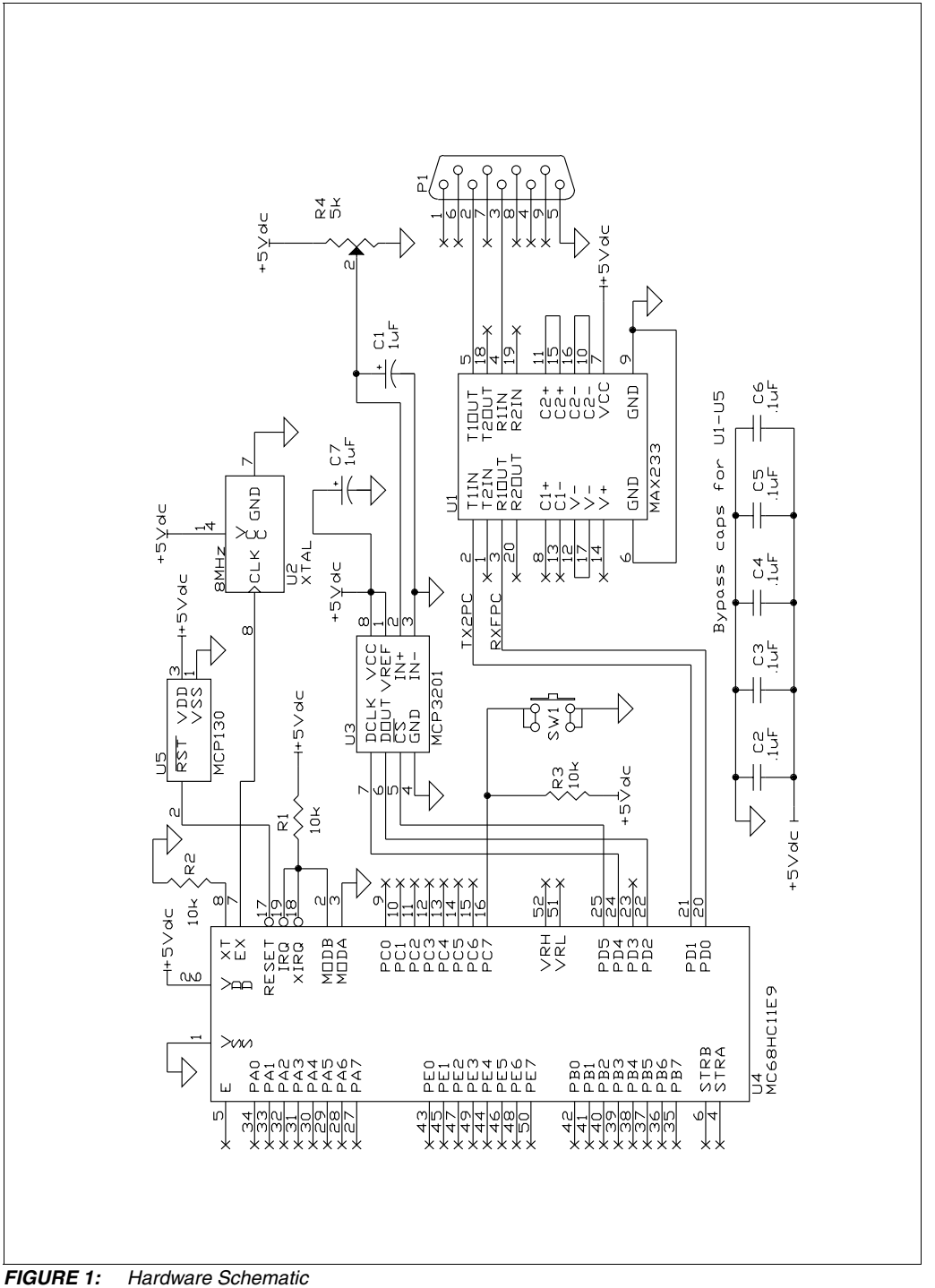


FIGURE 1: *Hardware Schematic*

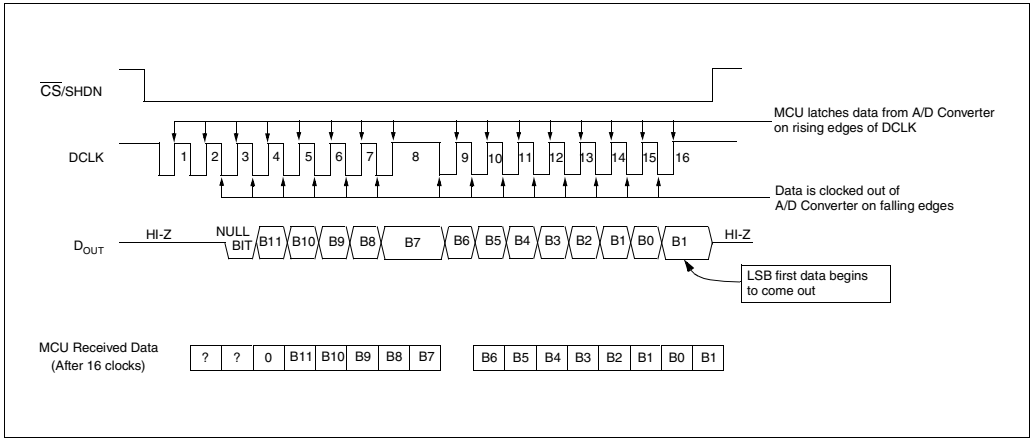


FIGURE 2: *SPI Communication using 8-bit segments (Mode 1,1: SCLK idles high).*

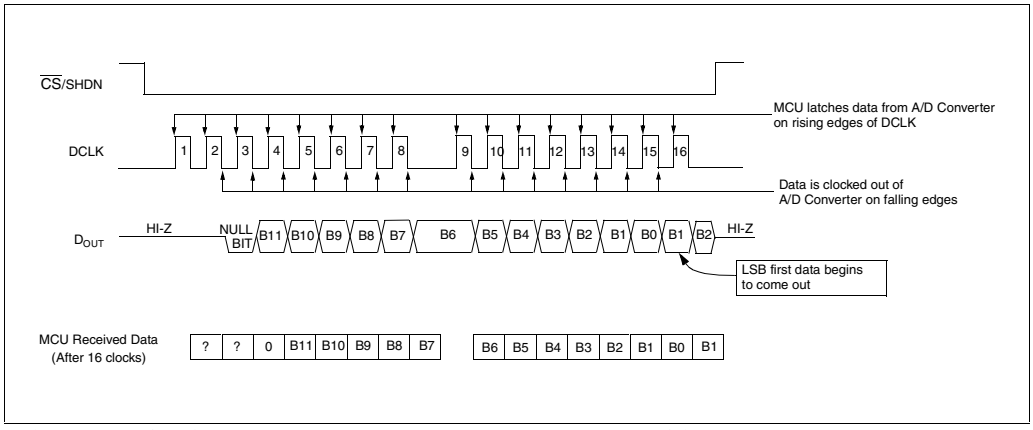


FIGURE 3: *SPI Communication using 8-bit segments (Mode 0,0: SCLK idles low).*

As the analog input signal is applied to the IN+ and IN- inputs, it is ratioed to the V_{REF} input for conversion scaling.

$$\textit{Digital output code} = \frac{V_{IN} \times F.S.}{V_{REF}}$$

Where:

V_{IN} = analog input voltage V(IN+) - V(IN–)

V_{REF} = reference voltage

F.S. = full scale = 4096

V_{REF} can be sourced directly from V_{DD} or can be supplied by an external reference. In either configuration, the V_{REF} source must be evaluated for noise contributions during the conversion. The voltage reference input of the MCP3201 ranges from 250mV to $5V_{DC}$, which approximately translates to a corresponding LSB size from 61µV to 1.22mV per bit.

$$1.22mV = \frac{5V_{DC}}{2^{12}\ bits}$$

For this simple application, the MCP3201 voltage reference input is tied to $5V_{DC}$. This translates to a 1.22mV/bit resolution for the A/D Converter module. The voltage input to the MCP3201 is implemented with a multi-turn potentiometer. The output voltage range of this passive driver is approximately $0V_{DC}$ to $5V_{DC}$.

Finally, a simple RS-232 interface is implemented using the USART peripheral of the microcontroller and a MAX233 transceiver IC. The USART transmits the captured A/D Converter binary value, both in ASCII and Decimal, to the PC terminal at 9600 baud.

As with all applications which require moderate to high performance A/D Converter operation, proper grounding and layout techniques are essential in achieving optimal performance. Proper power supply decoupling and input signal and V_{REF} parameters must be considered for noise contributions.

SOURCE CODE DESCRIPTION

The code written for this simple application performs five main functions:

1. Controller Initialization.
2. A/D Converter Conversion.
3. Conversion to ASCII.
4. Conversion to Decimal.
5. Transmit ASCII and Decimal to PC for display.

Upon power-up the microcontroller Port pins, USART peripheral and SSP module are initialized. The default microcontroller SPI bus mode is 1,1. The bus mode can be changed on power-up via SW1. If SW1 is depressed on power-up then PORTC:<7> is pulled low. The initialization code checks this pin state and if a logic low SPI bus mode 0,0 is selected. Likewise if PORTC:<7> is a logic high, (SW1 is not depressed), then mode 1,1 is the operational bus state. Once the initialization code is executed, the main code loop is entered and executed continuously.

After asserting PORTD:<5> the SPDR register is written to for initiating a SPI bus cycle. When the SPI cycle is complete, (SPIF flag is set to logic 1), the received data is read from the SPDR register and written to the RAM variable `RESULT_MSB`. The SPDR register is again written to, which initiates a SPI bus cycle, and the second 8-bits are received and written to the RAM variable `RESULT_LSB`. Here the composite result, located in variables `RESULT_MSB` and `RESULT_LSB` is right adjusted one bit location. The $\overline{CS}$/SHDN is then negated and the MCP3201 enters into the shutdown mode.

Next the `hex_to_ascii` and `hex_to_decimal` routines are called and executed. Then the `display_conversion` routine is executed which sends the data to the USART for transmission to the PC for display.

REFERENCES

Williams, Jim, “Analog Circuit Design,” Butterworth-Heinemann

Baker, Bonnie, “Layout Tips for 12-bit A/D Converter Applications,” *AN688, Microchip Technology Inc.*

MCP3201 12-bit A/D Converter with SPI Serial Interface, *Microchip Technology, Document DS21290B, 1999.*

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX A: SOURCE CODE

```
MCP3201.SRC Assembled with IASM 07/01/1999  11:30  PAGE 1
Interfacing Microchip MCP3201 ADC to Motorola MC68HC11E9-Based Microcontroller


                          1
                          2
                          3  ****************************************************************
                          4  *
                          5  *    Filename:      MCP3201.src
                          6  *    Date:          07/01/99
                          7  *
                          8  *    File Version:  1.00
                          9  *
                         10  *    Author:        Richard L. Fischer
                         11  *                   Microchip Technology Inc.
                         12  *
                         13  ****************************************************************
                         14  *
                         15  *
                         16  * This code demonstrates how the Microchip MCP3201 Analog-to-Digital
                         17  * Converter (ADC) is interfaced to the Synchronous Serial Peripheral
                         18  * (SSP) of the MC68HC11E9 microcontroller. The interface uses two
                         19  * Serial Peripheral Interface (SPI) lines (SCK, MISO) on the
                         20  * 68HC11E9 Microcontroller for the clock (SCK) and data in (MISO).
                         21  * A chip select (CS) to the MCP3201 is generated with a general
                         22  * purpose port line (PD5). The MC68HC11E9 is placed into the master
                         23  * mode which allows use of the port line PD5 for the CS control
                         24  * signal. The simple application uses Mode 00 or Mode 11 to the
                         25  * define bus clock polarity and phase.
                         26  *
                         27  * SPI bus mode 1,1 is the default mode of operation upon power-up.
                         28  * If SPI bus Mode 0,0 is desired, cycle off power, depress and
                         29  * hold SW1, cycle on power then release SW1. SPI bus Mode 0,0 is
                         30  * now the operational mode.
                         31  *
                         32  * For this application, the SPI data rate is set to one eighth of
                         33  * the microcontroller clock frequency. The MC68HC11E9 device clock
                         34  * frequency used for this application is 8MHz. This translates to
                         35  * an ADC throughput of 62.5kHz. In order to obtain the maximum
                         36  * throughput (100kHz) from the MCP3201 ADC the M68HC11E9 must be
                         37  * clocked at 12.8Mhz.
                         38  *
                         39  *
                         40  ****************************************************************
                         41
                         42
                         43  ***********************************************
                         44  *       MICROCONTROLLER RELATED EQUATES       *
                         45  ***********************************************
                         46
0000                     47  REGBASE   EQU  $1000     ; Register Base Address
0000                     48  PACTL     EQU  $26       ; PortA bit7 control
0000                     49  PORTA     EQU  $00       ; PortA Address
0000                     50  PORTB     EQU  $04       ; PortB Address
0000                     51  DDRC      EQU  $07       ; PortD Data Direction Register
0000                     52  PORTC     EQU  $03       ; PortC Address
0000                     53  DDRD      EQU  $09       ; PortD Data Direction Register
0000                     54  PORTD     EQU  $08       ; PortD Address
                         55
0000                     56  SPCR      EQU  $28       ; SPI Control Register
0000                     57  SPSR      EQU  $29       ; SPI Status Register
```



```
0000                    58  SPDR      EQU  $2A       ; SPI Data Register
                        59
0000                    60  BAUD      EQU  $2B       ; Baud Rate Register
0000                    61  SCCR2     EQU  $2D       ; SCI Control Register 2
0000                    62  SCSR      EQU  $2E       ; SCI Status Register
0000                    63  SCDR      EQU  $2F       ; SCI Data Register
                        64
                        65
0000                    66  MASKCS    EQU  $20       ; Mask for Chip Select Bit
                        67
                        68
                        69  ***********************************************
                        70  *       INTERNAL RAM USAGE                    *
                        71  ***********************************************
                        72
0000                    73            ORG  $0000     ; Internal RAM
                        74
0000                    75  RESULT_MSB     RMB  1    ; Converted MSB
0001                    76  RESULT_LSB     RMB  1    ; Converted LSB
                        77
0002                    78  THOUS          RMB  1    ; Variable for ASCII thousands
0003                    79  HUNDS          RMB  1    ; Variable for ASCII hundreds
0004                    80  TENS           RMB  1    ; Variable for ASCII tens
0005                    81  ONES           RMB  1    ; Variable for ASCII ones
                        82
0006                    83  DISPLAY_BUFF   RMB  1F   ; Buffer string for display
                        84
                        85
                        86
                        87  ***********************************************
                        88  *       INTERNAL MEMORY EQUATES               *
                        89  ***********************************************
                        90
0025                    91  EEPMBEG   EQU  $B600     ; EEPROM begin
0025                    92  EEPMEND   EQU  $B7FF     ; EEPROM end
                        93
0025                    94  EPRMBEG   EQU  $D000     ; EPROM begin
0025                    95  EPRMEND   EQU  $FFFF     ; EPROM end
                        96
0025                    97  RAMBEG    EQU  $0000     ; RAM begin
0025                    98  RAMEND    EQU  $01FF     ; RAM end
                        99
                       100
                       101
                       102  ****************************************************************
                       103
D000                   104            ORG   EPRMBEG           ; Beginning of code
                       105
D000 [03] 8E01FF       106  start     LDS   #RAMEND           ; Initialize Stack Pointer
D003 [06] 7F0000       107            CLR   RESULT_MSB        ; Clear ram variables
D006 [06] 7F0001       108            CLR   RESULT_LSB        ; Clear ram variables
                       109
D009 [03] CE0006       110            LDX   #DISPLAY_BUFF     ; Initialize X pointer to RAM
D00C [04] 18CED12D     111            LDY   #DISPLAY_IMAGE    ; Initialize Y pointer to ROM
D010 [05] 18A600       112  copy      LDAA  0,Y               ; Read from ROM
D013 [04] A700         113            STAA  0,X               ; Save into RAM
D015 [03] 08           114            INX                     ; Point to next RAM location
D016 [04] 1808         115            INY                     ; Point to next ROM location
D018 [04] 8C0026       116            CPX   #DISPLAY_BUFF+20  ; Copy display string complete?
D01B [03] 26F3         117            BNE   copy              ; No, so copy some more
                       118
                       119
D01D [03] CE1000       120            LDX   #REGBASE          ; Initialize X Index pointer
                       121
D020 [07] 1D00FF       122            BCLR  PORTA,X,$FF       ; Set PortA outputs low
D023 [07] 1C2680       123            BSET  PACTL,X,$80       ; Set PortA bit7 to output
```

```
D026 [07] 1D04FF     124            BCLR  PORTB,X,$FF       ; Set PortB outputs low
D029 [07] 1C077F     125            BSET  DDRC,X,$7F        ; Set PortC for outputs
                     126                                    ; except for pin7
D02C [07] 1D03FF     127            BCLR  PORTC,X,$FF       ; Set PortC outputs low
D02F [07] 1C0937     128            BSET  DDRD,X,$37        ; Make all PortD pins outputs
                     129                                    ; except PD3
D032 [02] 8620       130            LDAA  #$20              ;
D034 [04] A708       131            STAA  PORTD,X           ; Set CS high and all other
                     132
D036 [07] 1C2D08     133            BSET  SCCR2,X,$08       ; Enable SCI TX
D039 [02] 8630       134            LDAA  #$30              ; Initialize SCI for 9600
D03B [04] A72B       135            STAA  BAUD,X            ; baud at 8Mhz
                     136
D03D [07] 1E038006   137            BRSET PORTC,X,$80,m11   ; Branch to Mode 11 if pin high
D041 [02] 8650       138            LDAA  #$50              ; Else, Mode 00
D043 [04] A728       139            STAA  SPCR,X            ; Initialize SPI control reg
D045 [03] 2004       140            BRA   start_conversion  ; Go start conversion
                     141
D047 [02] 865C       142  m11       LDAA  #$5C              ; Mode 11
D049 [04] A728       143            STAA  SPCR,X            ; Initialize SPI control reg
                     144
                     145
                     146
                     147  ********************************************************
                     148  *
                     149  * The routines to follow perform 3 repetitive functions:
                     150  *
                     151  *   1. Initiate an ADC conversion sequence
                     152  *   2. Convert acquired data for display
                     153  *   2. Transmit converted data to PC
                     154  *
                     155  ********************************************************
                     156
                     157  start_conversion
                     158
D04B [07] 1D0820     159            BCLR  PORTD,X,MASKCS    ; Assert CS to ADC
                     160
D04E [06] 6F2A       161            CLR   SPDR,X            ; Initiate SPI bus cycle
D050 [07] 1F2980FC   162  rd1       BRCLR SPSR,X,$80,rd1    ; Wait until cycle completes
D054 [04] A62A       163            LDAA  SPDR,X            ; Read data and clear flag (SPIF)
D056 [03] 9700       164            STAA  RESULT_MSB        ; Save off upper bits
                     165
D058 [06] 6F2A       166            CLR   SPDR,X            ; Initiate SPI bus cycle
D05A [07] 1F2980FC   167  rd2       BRCLR SPSR,X,$80,rd2    ; Wait until cycle completes
                     168
D05E [07] 1C0820     169            BSET  PORTD,X,MASKCS    ; Negate CS, place ADC in shutdown
D061 [04] A62A       170            LDAA  SPDR,X            ; Read data and clear flag (SPIF)
D063 [03] 9701       171            STAA  RESULT_LSB        ; Save off lower bits
                     172
D065 [06] 760000     173            ROR   RESULT_MSB        ; Move bit 0 into carry
D068 [06] 760001     174            ROR   RESULT_LSB        ; Rotate MSB bit0 into LSB bit7
D06B [06] 1500F0     175            BCLR  RESULT_MSB,$F0    ; Mask out upper bits of MSB
                     176
D06E [06] BDD093     177            JSR   hex_to_ascii      ; Convert to ASCII for display
D071 [06] BDD0D3     178            JSR   hex_to_decimal    ; Convert to decimal for display
                     179
                     180
                     181
                     182  *****     ROUTINE FOR DISPLAYING CONVERTED DATA  ***********
                     183
                     184  display_conversion
                     185
D074 [04] 18CE0006   186            LDY   #DISPLAY_BUFF     ; Initialize Y pointer
D078 [05] 18A600     187  tx_loop   LDAA  0,Y               ; Retrieve MSB upper nibble
                     188
D07B [04] A72F       189            STAA  SCDR,X            ; Initiate SCI Transmit
```

```
D07D [07] 1F2E40FC   190  wtx       BRCLR SCSR,X,$40,wtx    ; Wait until cycle completes
D081 [04] 1808       191            INY                     ; Increment Y pointer
D083 [05] 188C0026   192            CPY   #DISPLAY_BUFF+20  ; All characters sent?
D087 [03] 26EF       193            BNE   tx_loop           ; No, so send more characters
                     194
                     195
D089 [04] 18CE0000   196            LDY #$0000              ; Approximately 485mS delay
D08D [04] 1809       197  decy      DEY                     ; Decrement counter
D08F [03] 26FC       198            BNE decy                ; Stay in loop until Y=0
                     199
D091 [03] 20B8       200            BRA start_conversion    ; Stay in main loop
                     201
                     202
                     203
                     204  *****     ROUTINE FOR CONVERTING TO ASCII  *****************
                     205
                     206  hex_to_ascii
                     207
D093 [04] 3C         208            PSHX                    ; Save off X Index register
D094 [03] CE0000     209            LDX   #RESULT_MSB       ; Initialize X
D097 [04] 18CE000E   210            LDY   #DISPLAY_BUFF+8   ; Initialize Y
D09B [04] A600       211  next      LDAA  0,X               ; Get Result_MSB data
D09D [03] 36         212            PSHA                    ; Save ACCA
D09E [02] 84F0       213            ANDA  #$F0              ; Mask out lower nibble
D0A0 [02] 44         214            LSRA                    ; Shift upper nibble
D0A1 [02] 44         215            LSRA                    ; into lower nibble
D0A2 [02] 44         216            LSRA                    ;
D0A3 [02] 44         217            LSRA                    ;
D0A4 [02] 8109       218            CMPA  #$09              ; Is value a number ?
D0A6 [03] 2F07       219            BLE   numb_1            ; Yes, make it 0-9 ($30 - $39)
D0A8 [02] 8B37       220            ADDA  #$37              ; No, make it a letter (A - F)
D0AA [05] 18A700     221            STAA  0,Y               ; Save ASCII letter
D0AD [03] 2005       222            BRA   do_lsb            ; Convert lower nibble
D0AF [02] 8B30       223  numb_1    ADDA  #$30              ;
D0B1 [05] 18A700     224            STAA  0,Y               ; Save ASCII number
                     225
D0B4 [04] 1808       226  do_lsb    INY                     ; Increment Y
D0B6 [04] 32         227            PULA                    ; Restore ACCA
D0B7 [02] 840F       228            ANDA  #$0F              ; Mask out upper nibble
D0B9 [02] 8109       229            CMPA  #$09              ; Is value a number ?
D0BB [03] 2F07       230            BLE   numb_2            ; Yes, make it 0-9 ($30 - $39)
D0BD [02] 8B37       231            ADDA  #$37              ; No, make it a letter (A - F)
D0BF [05] 18A700     232            STAA  0,Y               ; Save ASCII letter
D0C2 [03] 2005       233            BRA   next1             ;
                     234
D0C4 [02] 8B30       235  numb_2    ADDA  #$30              ;
D0C6 [05] 18A700     236            STAA  0,Y               ; Save ASCII number
                     237
D0C9 [03] 08         238  next1     INX                     ; Increment X
D0CA [04] 1808       239            INY                     ; Increment Y
D0CC [04] 8C0002     240            CPX   #RESULT_MSB+2     ; Converted all bytes?
D0CF [03] 26CA       241            BNE   next              ; No, so do some more
D0D1 [05] 38         242            PULX                    ; Else yes so restore X
D0D2 [05] 39         243            RTS                     ; Return from subroutine
                     244
                     245
                     246
                     247  *****     ROUTINE FOR CONVERTING TO DECIMAL  ***************
                     248
                     249  hex_to_decimal
                     250
D0D3 [04] DC00       251            LDD   RESULT_MSB        ; Initialize ACCA and ACCB
D0D5 [06] 7F0002     252            CLR   THOUS             ; Zero out THOUSANDS temp location
D0D8 [06] 7F0003     253            CLR   HUNDS             ; Zero out HUNDREDS temp location
D0DB [06] 7F0004     254            CLR   TENS              ; Zero out TENS temp location
D0DE [06] 7F0005     255            CLR   ONES              ; Zero out ONES temp location
```

```
                        256
D0E1 [05] 1A8303E7      257  ck_thous   CPD   #$03E7            ; ACCD >= 1000 or more?
D0E5 [03] 2508          258             BLO   ck_hunds          ; No, so go check hundreds
D0E7 [04] 8303E8        259             SUBD  #$03E8            ; Subtract 1000 from ACCD
D0EA [06] 7C0002        260             INC   THOUS             ; Increment THOUS
D0ED [03] 20F2          261             BRA   ck_thous          ; Go check for more
                        262
D0EF [05] 1A830063      263  ck_hunds   CPD   #$0063            ; ACCD >= 100 OR more?
D0F3 [03] 2508          264             BLO   ck_tens           ; No, so go check tens
D0F5 [04] 830064        265             SUBD  #$0064            ; Subtract 100 from ACCD
D0F8 [06] 7C0003        266             INC   HUNDS             ; Increment HUND
D0FB [03] 20F2          267             BRA   ck_hunds          ; Go check for more
                        268
D0FD [02] C10A          269  ck_tens    CMPB  #$0A              ; No, ACCB >= 10 or more?
D0FF [03] 2507          270             BLO   do_ones           ; No, finish up with ONES
D101 [02] C00A          271             SUBB  #$0A              ; Subtract another 10 from B
D103 [06] 7C0004        272             INC   TENS              ; Bump "TENS"
D106 [03] 20F5          273             BRA   ck_tens           ; Loop again
                        274
D108 [02] CB30          275  do_ones    ADDB  #$30              ; Convert ONES to ASCII
D10A [04] 18CE001E      276             LDY   #DISPLAY_BUFF+18  ; Initialize Y pointer into buffer
D10E [03] 9602          277             LDAA  THOUS             ;
D110 [02] 8B30          278             ADDA  #$30              ; Convert THOUS to ASCII
D112 [05] 18A700        279             STAA  0,Y               ; Save it
D115 [04] 1808          280             INY                     ; Increment Y
D117 [03] 9603          281             LDAA  HUNDS             ;
D119 [02] 8B30          282             ADDA  #$30              ; Convert HUNDS to ASCII
D11B [05] 18A700        283             STAA  0,Y               ; Save it
D11E [04] 1808          284             INY                     ; Increment Y
D120 [03] 9604          285             LDAA  TENS              ;
D122 [02] 8B30          286             ADDA  #$30              ; Convert TENS to ASCII
D124 [05] 18A700        287             STAA  0,Y               ; Save it
D127 [04] 1808          288             INY                     ; Increment Y
D129 [05] 18E700        289             STAB  0,Y               ; Save conversion into string
D12C [05] 39            290             RTS                     ; Return from subroutine
                        291
                        292
                        293
D12D      4865782D      294  DISPLAY_IMAGE   DB  'Hex-> 0x     : Decimal->     ',0D,0A
          3E203078
          20202020
          203A2044
          6563696D
          616C2D3E
          20202020
          200D0A
                        295
                        296
                        297  ***********************************************
                        298  *       INTERRUPT VECTORS                     *
                        299  ***********************************************
                        300
FFD6                    301         ORG  $FFD6      ; VECTORS
                        302
FFD6      D000          303         DW   start      ; SCI Serial System - RIE, TIE, TCIE, ILIE
FFD8      D000          304         DW   start      ; SPI Serial Transfer Complete - SPIE
FFDA      D000          305         DW   start      ; Pulse Accumulator Input Edge - PAII
FFDC      D000          306         DW   start      ; Pulse Accumulator Overflow - PAOVI
FFDE      D000          307         DW   start      ; Timer Overflow - TOI
FFE0      D000          308         DW   start      ; Timer Input Capture 4/Output Compare 5 - I4/O5I
FFE2      D000          309         DW   start      ; Timer Output Compare 4 - OC4I
FFE4      D000          310         DW   start      ; Timer Output Compare 3 - OC3I
FFE6      D000          311         DW   start      ; Timer Output Compare 2 - OC2I
FFE8      D000          312         DW   start      ; Timer Output Compare 1 - OC1I
FFEA      D000          313         DW   start      ; Timer Input Capture 3 - IC3I
FFEC      D000          314         DW   start      ; Timer Input Capture 2 - IC2I
FFEE      D000          315         DW   start      ; Timer Input Capture 1 -IC1I
```

```
 FFF0      D000        316        DW   start     ; Real Time Interrupt - RTII
 FFF2      D000        317        DW   start     ; IRQ (External Pin)
 FFF4      D000        318        DW   start     ; XIRQ Pin
 FFF6      D000        319        DW   start     ; Software Interrupt
 FFF8      D000        320        DW   start     ; Illegal Opcode Trap
 FFFA      D000        321        DW   start     ; COP Failure
 FFFC      D000        322        DW   start     ; Clock Monitor Fail
 FFFE      D000        323        DW   start     ; Reset
                       324
 0000                  325        END
                       326
                       327
                       328

 Symbol Table

BAUD              002B
CK_HUNDS          D0EF
CK_TENS           D0FD
CK_THOUS          D0E1
COPY              D010
DDRC              0007
DDRD              0009
DECY              D08D
DISPLAY_BUFF      0006
DISPLAY_CONVERSI  D074
DISPLAY_IMAGE     D12D
DO_LSB            D0B4
DO_ONES           D108
EEPMBEG           B600
EEPMEND           B7FF
EPRMBEG           D000
EPRMEND           FFFF
HEX_TO_ASCII      D093
HEX_TO_DECIMAL    D0D3
HUNDS             0003
M11               D047
MASKCS            0020
NEXT              D09B
NEXT1             D0C9
NUMB_1            D0AF
NUMB_2            D0C4
ONES              0005
PACTL             0026
PORTA             0000
PORTB             0004
PORTC             0003
PORTD             0008
RAMBEG            0000
RAMEND            01FF
RD1               D050
RD2               D05A
REGBASE           1000
RESULT_LSB        0001
RESULT_MSB        0000
SCCR2             002D
SCDR              002F
SCSR              002E
SPCR              0028
SPDR              002A
SPSR              0029
START             D000
START_CONVERSION  D04B
TENS              0004
THOUS             0002
TX_LOOP           D078
WTX               D07D
```

AN713

Controller Area Network (CAN) Basics

Author: *Keith Pazul*
Microchip Technology Inc.

INTRODUCTION

Controller Area Network (CAN) was initially created by German automotive system supplier Robert Bosch in the mid-1980s for automotive applications as a method for enabling robust serial communication. The goal was to make automobiles more reliable, safe and fuel-efficient while decreasing wiring harness weight and complexity. Since its inception, the CAN protocol has gained widespread popularity in industrial automation and automotive/truck applications. Other markets where networked solutions can bring attractive benefits like medical equipment, test equipment and mobile machines are also starting to utilize the benefits of CAN. The goal of this application note is to explain some of the basics of CAN and show the benefits of choosing CAN for embedded systems networked applications.

CAN OVERVIEW

Most network applications follow a layered approach to system implementation. This systematic approach enables interoperability between products from different manufacturers. A standard was created by the International Standards Organization (ISO) as a template to follow for this layered approach. It is called the ISO Open Systems Interconnection (OSI) Network Layering Reference Model and is shown in Figure 1 for reference.

The CAN protocol itself implements most of the lower two layers of this reference model. The communication medium portion of the model was purposely left out of the Bosch CAN specification to enable system designers to adapt and optimize the communication protocol on multiple media for maximum flexibility (twisted pair, single wire, optically isolated, RF, IR, etc.). With this flexibility, however, comes the possibility of interoperability concerns.

To ease some of these concerns, the International Standards Organization and Society of Automotive Engineers (SAE) have defined some protocols based on CAN that include the Media Dependant Interface definition such that all of the lower two layers are specified. ISO11898 is a standard for high-speed applications, ISO11519 is a standard for low-speed applications, and J1939 (from SAE) is targeted for truck and bus applications. All three of these protocols specify a 5V differential electrical bus as the physical interface.

The rest of the layers of the ISO/OSI protocol stack are left to be implemented by the system software developer. Higher Layer Protocols (HLPs) are generally used to implement the upper five layers of the OSI Reference Model.

HLPs are used to:

1) standardize startup procedures including bit rates used,
2) distribute addresses among participating nodes or types of messages,
3) determine the structure of the messages, and
4) provide system-level error handling routines.

This is by no means a full list of the functions HLPs perform, however it does describe some of their basic functionality.

CAN PROTOCOL BASICS

Carrier Sense Multiple Access with Collision Detection (CSMA/CD)

The CAN communication protocol is a CSMA/CD protocol. The CSMA stands for Carrier Sense Multiple Access. What this means is that every node on the network must monitor the bus for a period of no activity before trying to send a message on the bus (Carrier Sense). Also, once this period of no activity occurs, every node on the bus has an equal opportunity to transmit a message (Multiple Access). The CD stands for Collision Detection. If two nodes on the network start transmitting at the same time, the nodes will detect the 'collision' and take the appropriate action. In CAN protocol, a non-destructive bitwise arbitration method is utilized. This means that messages remain intact after arbitration is completed even if collisions are detected. All of this arbitration takes place without corruption or delay of the higher priority message.

There are a couple of things that are required to support non-destructive bitwise arbitration. First, logic states need to be defined as dominant or recessive. Second, the transmitting node must monitor the state of the bus to see if the logic state it is trying to send actually appears on the bus. CAN defines a logic bit 0 as a dominant bit and a logic bit 1 as a recessive bit.

A dominant bit state will always win arbitration over a recessive bit state, therefore the lower the value in the Message Identifier (the field used in the message arbitration process), the higher the priority of the message. As an example, suppose two nodes are trying to transmit a message at the same time. Each node will monitor the bus to make sure the bit that it is trying to send actually appears on the bus. The lower priority message will at some point try to send a recessive bit and the monitored state on the bus will be a dominant. At that point this node loses arbitration and immediately stops transmitting. The higher priority message will continue until completion and the node that lost arbitration will wait for the next period of no activity on the bus and try to transmit its message again.

Message-Based Communication

CAN protocol is a message-based protocol, not an address based protocol. This means that messages are not transmitted from one node to another node based on addresses. Embedded in the CAN message itself is the priority and the contents of the data being transmitted. All nodes in the system receive every message transmitted on the bus (and will acknowledge if the message was properly received). It is up to each node in the system to decide whether the message received should be immediately discarded or kept to be processed. A single message can be destined for one particular node to receive, or many nodes based on the way the network and system are designed.

For example, an automotive airbag sensor can be connected via CAN to a safety system router node only. This router node takes in other safety system information and routes it to all other nodes on the safety system network. Then all the other nodes on the safety system network can receive the latest airbag sensor information from the router at the same time, acknowledge if the message was received properly, and decide whether to utilize this information or discard it.

Another useful feature built into the CAN protocol is the ability for a node to request information from other nodes. This is called a Remote Transmit Request (RTR). This is different from the example in the previous paragraph because instead of waiting for information to be sent by a particular node, this node specifically requests data to be sent to it.

For example, a safety system in a car gets frequent updates from critical sensors like the airbags, but it may not receive frequent updates from other sensors like the oil pressure sensor or the low battery sensor to make sure they are functioning properly. Periodically, the safety system can request data from these other sensors and perform a thorough safety system check. The system designer can utilize this feature to minimize network traffic while still maintaining the integrity of the network.

One additional benefit of this message-based protocol is that additional nodes can be added to the system without the necessity to reprogram all other nodes to recognize this addition. This new node will start receiving messages from the network and, based on the message ID, decide whether to process or discard the received information.

CAN Message Frame Description

CAN protocol defines four different types of messages (or Frames). The first and most common type of frame is a Data Frame. This is used when a node transmits information to any or all other nodes in the system. Second is a Remote Frame, which is basically a Data Frame with the RTR bit set to signify it is a Remote Transmit Request (see Figure 2 and Figure 3 for details on Data Frames). The other two frame types are for handling errors. One is called an Error Frame and one is called an Overload Frame. Error Frames are generated by nodes that detect any one of the many protocol errors defined by CAN. Overload errors are generated by nodes that require more time to process messages already received.

Data Frames consist of fields that provide additional information about the message as defined by the CAN specification. Embedded in the Data Frames are Arbitration Fields, Control Fields, Data Fields, CRC Fields, a 2-bit Acknowledge Field and an End of Frame.

The Arbitration Field is used to prioritize messages on the bus. Since the CAN protocol defines a logical 0 as the dominant state, the lower the number in the arbitration field, the higher priority the message has on the bus. The arbitration field consists of 12-bits (11 identifier bits and one RTR bit) or 32-bits (29 identifier bits, 1-bit to define the message as an extended data frame, an SRR bit which is unused, and an RTR bit), depending on whether Standard Frames or Extended Frames are being utilized. The current version of the CAN specification, version 2.0B, defines 29-bit identifiers and calls them Extended Frames. Previous versions of the CAN specification defined 11-bit identifiers which are called Standard Frames.

As described in the preceding section, the Remote Transmit Request (RTR) is used by a node when it requires information to be sent to it from another node. To accomplish an RTR, a Remote Frame is sent with the identifier of the required Data Frame. The RTR bit in the Arbitration Field is utilized to differentiate between a Remote Frame and a Data Frame. If the RTR bit is recessive, then the message is a Remote Frame. If the RTR bit is dominant, the message is a Data Frame.

The Control Field consists of six bits. The MSB is the IDE bit (signifies Extended Frame) which should be dominant for Standard Data Frames. This bit determines if the message is a Standard or Extended Frame. In Extended Frames, this bit is RB1 and it is reserved. The next bit is RB0 and it is also reserved. The four LSBs are the Data Length Code (DLC) bits. The Data Length Code bits determine how many data bytes are included in the message. It should be noted that a Remote Frame has no data field, regardless of the value of the DLC bits.

The Data Field consists of the number of data bytes described in the Data Length Code of the Control Field.

The CRC Field consists of a 15-bit CRC field and a CRC delimiter, and is used by receiving nodes to determine if transmission errors have occurred.

The Acknowledge Field is utilized to indicate if the message was received correctly. Any node that has correctly received the message, regardless of whether the node processes or discards the data, puts a dominant bit on the bus in the ACK Slot bit time (see Figure 2 or Figure 3 for the location of the ACK Slot bit time).

The last two message types are Error Frames and Overload Frames. When a node detects one of the many types of errors defined by the CAN protocol, an Error Frame occurs. Overload Frames tell the network that the node sending the Overload Frame is not ready to receive additional messages at this time, or that intermission has been violated. These errors will be discussed in more detail in the next section.

Fast, Robust Communication

Because CAN was initially designed for use in automobiles, a protocol that efficiently handled errors was critical if it was to gain market acceptance. With the release of version 2.0B of the CAN specification, the maximum communication rate was increased 8x over the version 1.0 specification to 1Mbit/sec. At this rate, even the most time-critical parameters can be transmitted serially without latency concerns. In addition to this, the CAN protocol has a comprehensive list of errors it can detect that ensures the integrity of messages.

CAN nodes have the ability to determine fault conditions and transition to different modes based on the severity of problems being encountered. They also have the ability to detect short disturbances from permanent failures and modify their functionality accordingly. CAN nodes can transition from functioning like a normal node (being able to transmit and receive messages normally), to shutting down completely (bus-off) based on the severity of the errors detected. This feature is called Fault Confinement. No faulty CAN node or nodes will be able to monopolize all of the bandwidth on the network because faults will be confined to the faulty nodes and these faulty nodes will shut off before bringing the network down. This is very powerful because Fault Confinement guarantees bandwidth for critical system information.

As discussed previously, there are five error conditions that are defined in the CAN protocol and three error states that a node can be in, based upon the type and number of error conditions detected. The following section describes each one in more detail.

Errors Detected

CRC Error

A 15-bit Cyclic Redundancy Check (CRC) value is calculated by the transmitting node and this 15-bit value is transmitted in the CRC field. All nodes on the network receive this message, calculate a CRC and verify that the CRC values match. If the values do not match, a CRC error occurs and an Error Frame is generated. Since at least one node did not properly receive the message, it is then resent after a proper intermission time.

Acknowledge Error

In the Acknowledge Field of a message, the transmitting node checks if the Acknowledge Slot (which it has sent as a recessive bit) contains a dominant bit. This dominant bit would acknowledge that at least one node correctly received the message. If this bit is recessive, then no node received the message properly. An Acknowledge Error has occurred. An Error Frame is then generated and the original message will be repeated after a proper intermission time.

Form Error

If any node detects a dominant bit in one of the following four segments of the message: End of Frame, Interframe Space, Acknowledge Delimiter or CRC Delimiter, the CAN protocol defines this to be a form violation and a Form Error is generated. The original message is then resent after a proper intermission time. (see Figure 2 and/or Figure 3 for where these segments lie in a CAN message).

Bit Error

A Bit Error occurs if a transmitter sends a dominant bit and detects a recessive bit, or if it sends a recessive bit and detects a dominant bit when monitoring the actual bus level and comparing it to the bit that it has just sent. In the case where the transmitter sends a recessive bit and a dominant bit is detected during the Arbitration Field or Acknowledge Slot, no Bit Error is generated because normal arbitration or acknowledgment is occurring. If a Bit Error is detected, an Error Frame is generated and the original message is resent after a proper intermission time.

Stuff Error

CAN protocol uses a Non-Return–to-Zero (NRZ) transmission method. This means that the bit level is placed on the bus for the entire bit time. CAN is also asynchronous, and bit stuffing is used to allow receiving nodes to synchronize by recovering clock information from the data stream. Receiving nodes synchronize on recessive to dominant transitions. If there are more than five bits of the same polarity in a row, CAN will automatically stuff an opposite polarity bit in the data stream. The receiving node(s) will use it for synchronization, but will ignore the stuff bit for data purposes. If, between the Start of Frame and the CRC Delimiter, six consecutive bits with the same polarity are detected, then the bit stuffing rule has been violated. A Stuff Error then occurs, an Error Frame is sent, and the message is repeated.

Error States

Detected errors are made public to all other nodes via Error Frames or Error Flags. The transmission of an erroneous message is aborted and the frame is repeated as soon as the message can again win arbitration on the network. Also, each node is in one of three error states, Error-Active, Error-Passive or Bus-Off.

Error-Active

An Error-Active node can actively take part in bus communication, including sending an active error flag, which consists of six consecutive dominant bits. The Error Flag actively violates the bit stuffing rule and causes all other nodes to send an Error Flag, called the Error Echo Flag, in response. An Active Error Flag, and the subsequent Error Echo Flag may cause as many as twelve consecutive dominant bits on the bus; six from the Active Error Flag, and zero up to six more from the Error Echo Flag depending upon when each node detects an error on the bus. A node is Error-Active when both the Transmit Error Counter (TEC) and the Receive Error Counter (REC) are below 128. Error-Active is the normal operational mode, allowing the node to transmit and receive without restrictions.

Error-Passive

A node becomes Error-Passive when either the Transmit Error Counter or Receive Error Counter exceeds 127. Error-Passive nodes are not permitted to transmit Active Error Flags on the bus, but instead, transmit Passive Error Flags which consist of six recessive bits. If the Error-Passive node is currently the only transmitter on the bus then the passive error flag will violate the bit stuffing rule and the receiving node(s) will respond with Error Flags of their own (either active or passive depending upon their own error state). If the Error-Passive node in question is not the only transmitter (i.e. during arbitration) or is a receiver, then the Passive Error Flag will have no effect on the bus due to the recessive nature of the error flag. When an Error-Passive node transmits a Passive Error Flag and detects a dominant bit, it must see the bus as being idle for eight additional bit times after an intermission before recognizing the bus as available. After this time, it will attempt to retransmit.

Bus-Off

A node goes into the Bus-Off state when the Transmit Error Counter is greater than 255 (receive errors can not cause a node to go Bus-Off). In this mode, the node can not send or receive messages, acknowledge messages, or transmit Error Frames of any kind. This is how Fault Confinement is achieved. There is a bus recovery sequence that is defined by the CAN protocol that allows a node that is Bus-Off to recover, return to Error-Active, and begin transmitting again if the fault condition is removed.

CONCLUSION

The CAN protocol was optimized for systems that need to transmit and receive relatively small amounts of information (as compared to Ethernet or USB, which are designed to move much larger blocks of data) reliably to any or all other nodes on the network. CSMA/CD allows every node to have an equal chance to gain access to the bus, and allows for smooth handling of collisions.

Since the protocol is message-based, not address based, all messages on the bus receive every message and acknowledge every message, regardless of whether in needs the data or not. This allows the bus to operate in node-to-node or multicast messaging formats without having to send different types of messages.

Fast, robust message transmission with fault confinement is also a big plus for CAN because faulty nodes will automatically drop off the bus not allowing any one node from bringing a network down. This effectively guarantees that bandwidth will always be available for critical messages to be transmitted. With all of these benefits built into the CAN protocol and its momentum in the automotive world, other markets will begin to see and implement CAN into their systems.

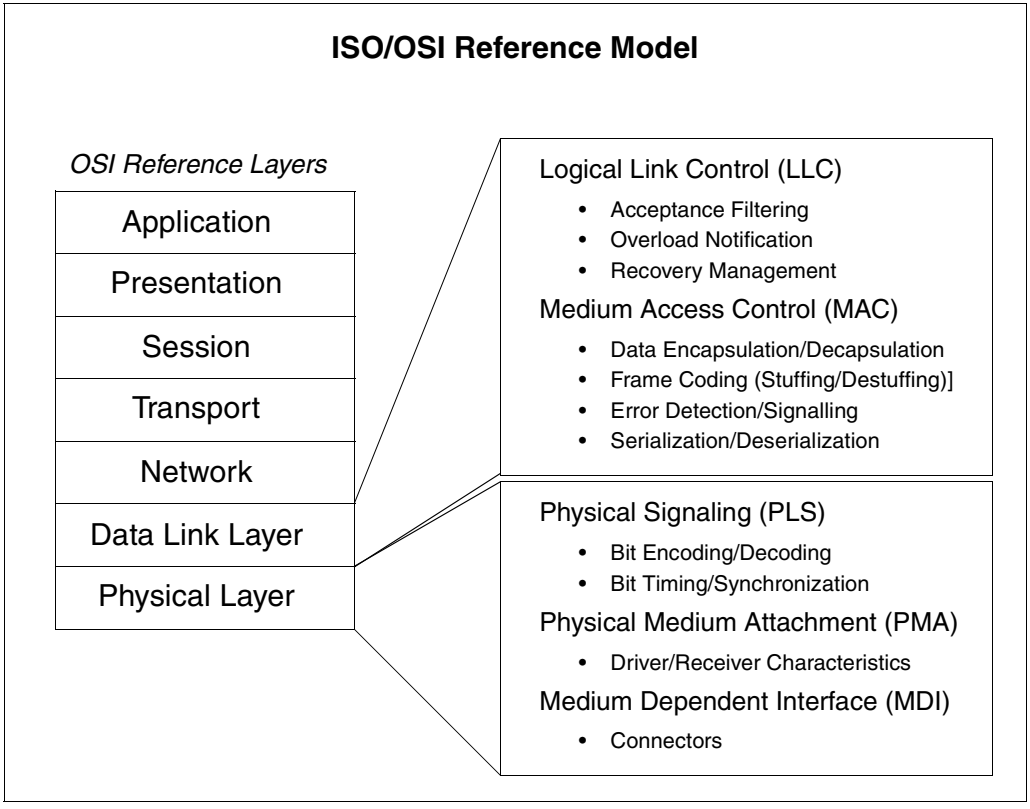


FIGURE 1: *ISO/OSI Reference Model*

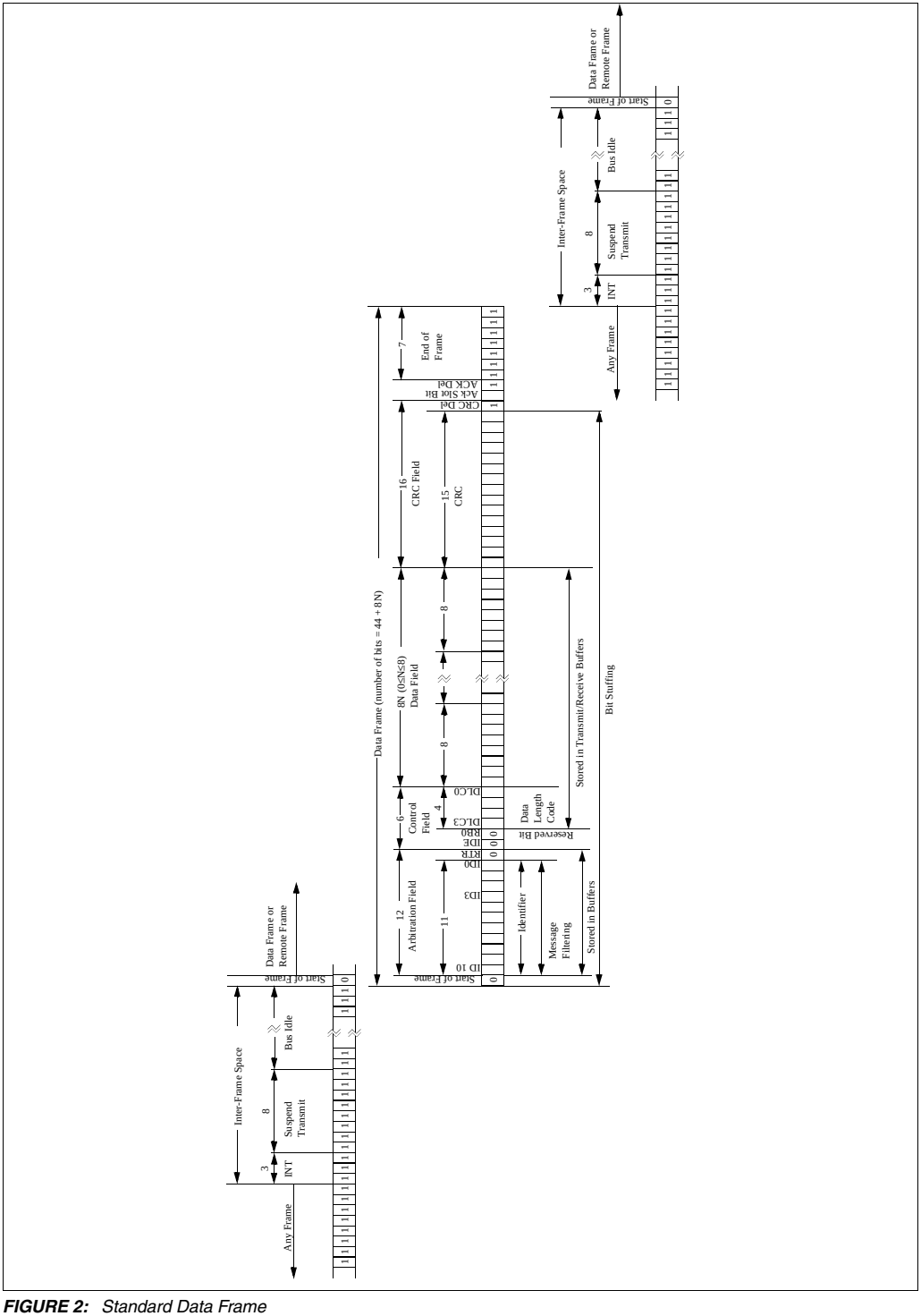


FIGURE 2: *Standard Data Frame*

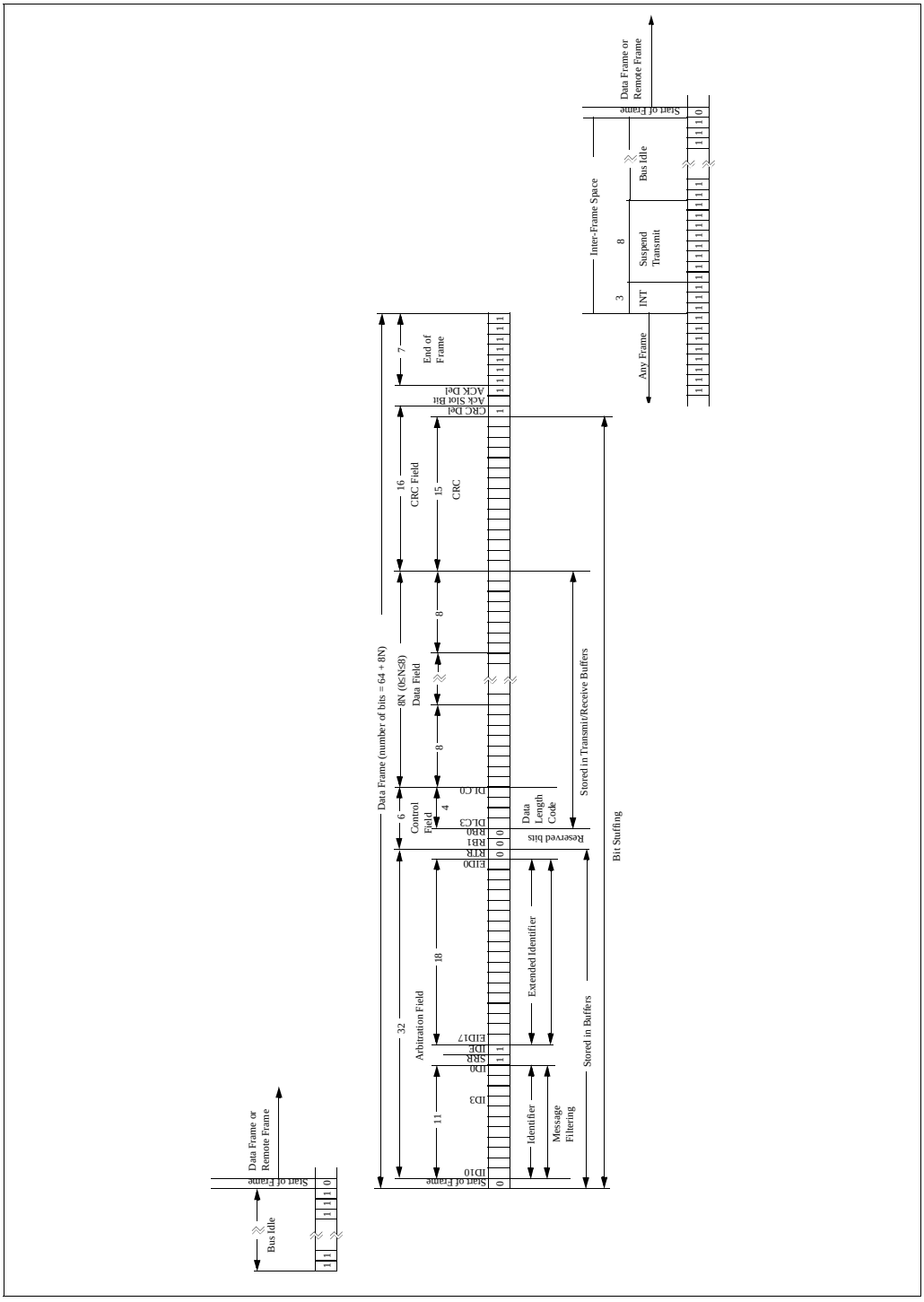


FIGURE 3: *Extended Data Frame*

NOTES:

AN717

Building a 10-bit Bridge Sensing Circuit using the PIC16C6XX and MCP601 Operational Amplifier

Author: *Bonnie C. Baker*
Microchip Technology Inc.

INTRODUCTION

Sensors that use Wheatstone bridge configurations, such as pressure sensors, load cells, or thermistors have a great deal of commonality when it comes to the signal conditioning circuit. This application note delves into the inner workings of the electronics of the signal conditioning path for sensors that use Wheatstone bridge configurations. Analog topics such as gain and filtering circuits will be explored. This discussion is complemented with digital issues such as digital filtering and digital manipulation techniques. Overall, this note's comprehensive investigation of hardware and firmware provides a practical solution including error correction in the data acquisition sensor system.

A sensor that is configured in a Wheatstone bridge typically supplies a low level, differential output signal. The application problem the designer is challenged with is to capture this small signal and eventually convert it to a digital format that gives an 8 to 12 bit representation of the signal.

The inexpensive design strategy shown in the block diagram in Figure 1 uses a low pass filter prior to digitization. The conversion from analog to digital is performed with the microcontroller (MCU). The MCU used in this circuit must have internal analog functions such as a voltage reference and a comparator. These internal analog building blocks are used to implement a first order modulator. This is combined with the MCU's computing power where a digital filter can be implemented.

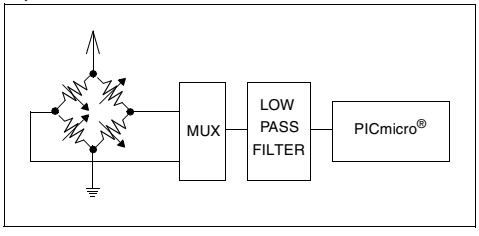


FIGURE 1: *The bridge sensor signal conditioning chain filters high frequency noise in the analog domain then immediately digitizes it with a microcontroller.*

BRIDGE SENSOR DATA ACQUISITION CIRCUIT IMPLEMENTATION

Figure 2 gives a detailed circuit for the block diagram shown in Figure 1. The analog portion of this circuit consists of the sensor (in this example a 1.2kΩ, 2mV/V load cell), an analog multiplexer, an amplifier and an R/C network. (A complete list of the load cell's specifications is given in Table 1.) An analog multiplexer is used to switch the two sensor outputs between a single ended signal path to the controller. The amplifier is configured as a buffer and used to isolate the sensor load from the R/C network. The R/C network implements the integrator function of a first order modulator. This network can also be used to adjust the input range to the MCU.

Rated Capacity	32 ounces (896 g)
Excitation	$5V_{DC}$ to $12V_{DC}$
Rated Output	2mV/V ±20%
Zero Balance	±0.3mV/V
Operating Temperature	-55 to 95°C
Compensated Temperature	-5 to 50°C
Zero Balance over Temperature	0.036% FS/°C
Output over Temperature	0.036% FS/°C
Resistance	1200Ω ±300Ω
Safe Overload	150%
Full-Scale Deflection	0.01" to 0.05"

TABLE 1: *Load Cell, LCL816G (Omega) Specifications.*

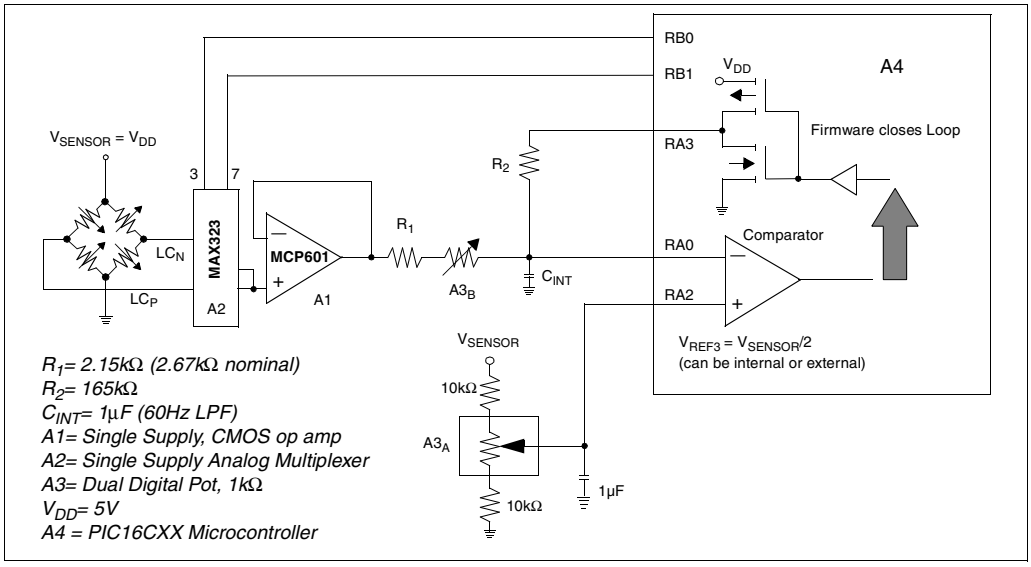


FIGURE 2: *The combination of an R/C network and the microcontroller's analog peripherals can be used to perform an A/D conversion function.*

In this circuit, the integrator function of the modulator is implemented with an external capacitor, C_{INT}. When RA3 of the PIC16C6XX is set high, the voltage at RA0 increases in magnitude. This occurs until the output of the comparator (CMCON<6>) is triggered low. At this point, the driver to the RA3 output is switched from high to low. Once this has transpired, the voltage at the input to the comparator (RA0) decreases. This occurs until the comparator is tripped high. At this point, RA3 is set high and the cycle repeats. While the modulator section of this circuit is cycling, two counters are used to keep track of the time and of the number of ones versus zeros that occur at the output of the comparator. The firmware flow chart for this conversion process is shown in Figure 3.

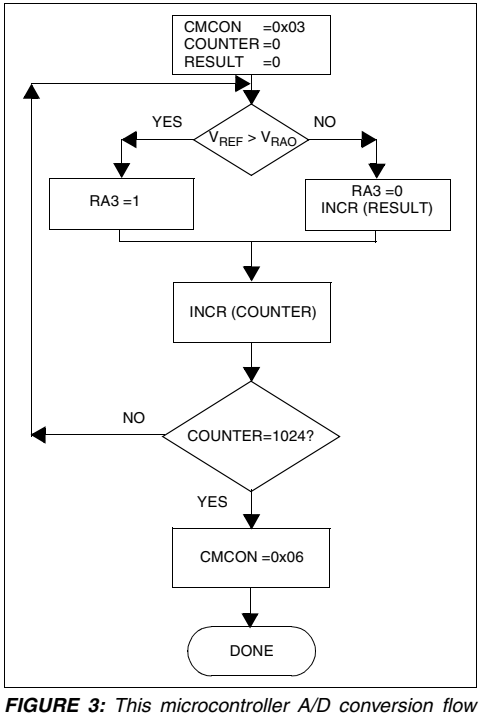


FIGURE 3: *This microcontroller A/D conversion flow chart is implemented with the circuit shown in Figure 2. Care should be taken to make the time that every cycle takes through the flow chart constant.*

After the timing counter goes to 1024 on one side of the Wheatstone bridge, the MCU switches the multiplexer (A2) to the leg of the other side of the sensor. With the voltage of the other leg of the sensor connected to the input of the amplifier, the controller cycles through another conversion for 1024 counts. The two results from these cycles are subtracted, giving the conversion results. This technique provides 10-bits of resolution with 9.9-bits of accuracy (rms).

The design equations for this circuit are:

$$V_{IN\ (CM)} = V_{REF3}$$

$$V_{IN\ (P\ to\ P)} = V_{RA3\ (P\ to\ P)}\ (R_1 / R_2)$$

with $V_{IN\ (CM)}$ approximates V_{DD} /2 or is equal to $(LC_P + LC_N)/2$,

where:

V_{REF3} is the voltage reference applied to the comparator's non-inverting input and equal to approximately V_{SENSOR} /2. If made external, this reference voltage can be used to adjust offset errors

$V_{IN\ (P\ to\ P)}$ is equal to $(LC_{P(MAX)} - LC_{N(MIN)})$ or $(LC_{P(MIN)} - LC_{N(MAX)})$ which equals the sensor full scale range and $V_{RA3\ (P\ to\ P)}$ is equal to $V_{RA3\ (MAX)} - V_{RA3\ (MIN)}$ or approximately V_{DD}

The system in this application note has been designed to have a full-scale input range to the comparator of ± 40.5mV. Given 9.9-bit (rms) accuracy, the LSB size is 84.7µV.

The transfer function of the percentage of ones counted versus input voltage is shown in Figure 4. In this diagram, both the duty cycle between ones and zeros as well as the pulse width is modulated.

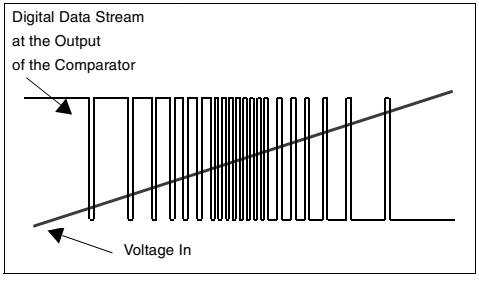


FIGURE 4: *The relationship between input voltage and the number of ones that are counted by the controller is shown conceptually with this diagram. At low input voltages, the output of the comparator produces very few zeros within the 1024 counts. At voltages in the center of the input range, the comparator quickly toggles between ones and zeros. At higher voltages, the comparator output produces mostly zeros and very few ones.*

ERROR SOURCES AND SYSTEM SOLUTIONS

A wheatstone bridge is designed to give a differential output rendering a small voltage that changes proportionally to the sensor's excitation, i.e. pressure or temperature, etc. The dominant types of errors that a sensor exhibits in its transfer function can be categorized as offset, gain, linearity, noise, and thermal. These sensors also produce other errors such as hysteresis, repeatability, stability and aging that are beyond the scope of this application note. An equal contributor to the overall system errors is the offset, gain, and linearity errors from the active components in the signal conditioning path.

OFFSET ERRORS

The offset error of a system can be mathematically described with a constant additive to the entire transfer function as shown in Figure 5.

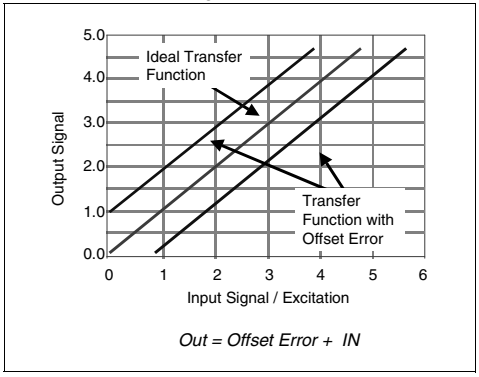


FIGURE 5: *The offset error of a system can be described graphically with a transfer function that has shifted along the x-axis.*

Typically, offset error is measured at a point where the input signal to the system is zero. This technique provides an output signal that is equal to the offset. This type of error can originate at the sensor or within the various components in the analog signal path. By definition, the offset error is repeatable and stable at a specified operating condition. If the operating conditions change, such as temperature, voltage excitation or current excitation, the offset error may also change.

The offset errors in this signal path come from the wheatstone bridge sensor, the operational amplifier (A1) offset, the port leakage current at RA0, the internal voltage reference (V_{REF3}) offset, the comparator offset, and the non-symmetrical output port of RA3. The only difference in the signal path of the two sensor outputs is the multiplexer channel, which interfaces directly with a high impedance CMOS operational amplifier. Otherwise, both sensor output signals are configured to travel down the same signal-conditioning path. Consequently, the conversion data taken from the positive leg

of the load cell sensor (LC_P) has the same offset and gain errors as the conversion data taken from the negative output of the load cell sensor (LP_N), with the exception of the bridge's offset error.

To accommodate these errors, the design equations for this circuit remains:

$$V_{IN\ (CM)} = V_{REF3}$$

$$V_{IN(P\ to\ P)} = V_{RA3\ (P\ to\ P)}\ (R_1 / R_2)$$

But, now the worst case variation of $V_{IN(P\ to\ P)}$ is equal to $(LC_{P(MAX)} - LC_{P(MIN)} + LC_{OFF} + A1_{OFF} + RA0_{OFF} + V_{REF3\text{-}OFF} + COMP_{OFF} + RA3_{OFF})$

where:

ΔLC_{OFF} is the maximum offset voltage that can be generated by the load cell bridge

$A1_{OFF}$ is the offset voltage of the operational amplifier

$RA0_{OFF}$ is the offset error caused by the leakage current of port RA0. This leakage current is specified at 1nA at room temperature and 0.5µA (max) over temperature. This leakage current causes a voltage drop across the parallel combination of R_1 and R_2

$V_{REF3\text{-}OFF}$ is the offset error of the internal voltage reference of the MCU or $V_{DD}/2$. This error can be reduced significantly with an external voltage reference

$COMP_{OFF}$ is the offset of the internal comparator of the MCU and

$RA3_{OFF}$ is caused by the inability of RA3 to go completely to the rails. It can be quantified by $RA3_{OFF} = ((V_{DD} - RA3_{HIGH}) - RA3_{LOW})/2$. This formula assumes $V_{REF3} = V_{DD} / 2$

The maximum magnitudes of these errors are summarized in Table 2.

Error Source	Offset Voltage over Temperature
Load Cell Bridge	±1.5mV in a 5V system
Op Amp	±2mV
Port Leakage, RA0	±1.3mV
Internal V_{REF}	±49mV
Comparator	±10mV
Output Port, RA3 (asymmetrical output swing)	5.5mV

TABLE 2: *Maximum offset errors over temperature for the circuit shown in Figure 3.*

Firmware Offset Calibration

The offset errors of the circuit can be calibrated in firmware. This is performed by subtracting the conversion code results of the positive leg of the sensor from the results of the negative leg of the sensor. The analog representation of the result of this calculation in firmware is:

$$V_{OUT} = LC_P + LC_{OFF} + A1_{OFF} + RA0_{OFF} + V_{REF3\text{-}OFF} + COMP_{OFF} + RA3_{OFF} - LC_N - A1_{OFF} - RA0_{OFF} - V_{REF3\text{-}OFF} - COMP_{OFF} - RA3_{OFF}$$

$$V_{OUT} = LC_P - LC_N + LC_{OFF}$$

This result illustrates the efficiency of using firmware to eliminate most of the offset errors, however, the trade-off for having offset adjustments performed by the MCU is dynamic range. In anticipation of these offset errors, the designer should increase the peak-to-peak analog input range of the conversion system. This will result in a conversion that has a wider dynamic range, consequently, lower accuracy.

To counteract this, the accuracy can be improved if more samples are taken in the conversion process. This technique will elongate the overall conversion time. Another technique that can be used is to perform a simple offset adjust in hardware which can be implemented with the digital potentiometer ($A3_A$).

Hardware Offset Calibration

Given the design equations for this circuit and the errors in Table 2, the total expected offset error over temperature for the electronics is ±69.3mV. With a sensor full-scale range of ±10mV, the dynamic range of the system would be ~7.9 times larger than the nominal, error free peak-to-peak range at the input of the comparator.

The 8-bit, 1kΩ digital potentiometer ($A3_A$) in Figure 2 is placed in series between the power supply (5V) with two 10kΩ, 1% resistors. This configuration provides a voltage reference range to the comparator of ±119mV centered around mid-supply (2.5V) with a resolution of 0.5mV. If an external reference is used with a ±0.5mV error range, the electronics will contribute ±20.8mV offset error over temperature. This changes the worst case full-scale peak-to-peak range of the system to ±30.8mV. This is only approximately three times larger than the nominal full-scale output (±10mV) of the sensor.

System Span (Gain) Errors

The span or gain of a system can be mathematically described as a constant, which is multiplied against the input signal. The magnitude of the span can easily be determined using the formula below:

Ideal Output = Input x Gain

Span error is the deviation of the span multiple from ideal and can be described with the formula below:

Actual Output = Input x Span (1 + Span Error)

Examples of transfer functions with span error are shown in Figure 6. The plots in Figure 6 do not have offset errors.

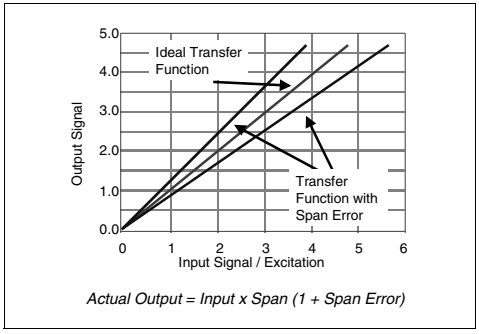


FIGURE 6: *Span or gain errors can be described graphically as a transfer function that rotates around the intercept of the x and y-axis.*

Firmware Span (Gain) Calibration

For this circuit, the span error of the sensor is less influential than the offset errors on the system. Span errors come from the Load Cell (±20%), the resistors (±1%), capacitor (±10%), and the ON resistance of the RA3 port (0.2%). This circuit can rely on firmware calibration with a reduction in the dynamic range of the system. The combination of the sensor error and the capacitor error increases the requirements on the input range to the modulator configuration.

Hardware Span (Gain) Calibration

Span errors can most effectively be removed in the analog domain. For instance, the span error of the sensor can be adjusted with the sensor's excitation voltage. As a trade-off for this adjustment strategy, the common mode voltage of the sensor is changed, creating offset errors with respect to the reference voltage (V_{REF3}) of the comparator. This problem can be alleviated by making the voltage reference ratiometric to the sensor excitation source. Span errors can also be adjusted with either R_1 or R_2. In the circuit in Figure 2, the other half of the dual, 1kΩ digital potentiometer ($A3_B$) is configured to perform this function. This type of adjustment does not change the offset error of the system. Finally, span errors can be corrected with changes to the integration capacitor. Of all of the span adjustments, this one is the most awkward to implement.

SYSTEM LINEARITY ERRORS

Linearity error differs from offset or span errors in that it has a unique affect on each individual code of the digitizing system. Linearity errors are defined as the deviation of the transfer function from a straight line. Some engineers define this error using a line that stretches between the end points of the transfer curve while others define it using a line that is calculated using a "best fit" algorithm. In either case, the linearity errors can cause significant errors in translating the sensor input (pressure, temperature, etc.) to digital code.

Linearity errors come in many forms as shown in Figure 7. Sometimes the linearity error of a system can be characterized with a multi-order polynomial, but more typically this error is difficult to predict from system to system, in which case, firmware piecewise linearization methods are usually used.

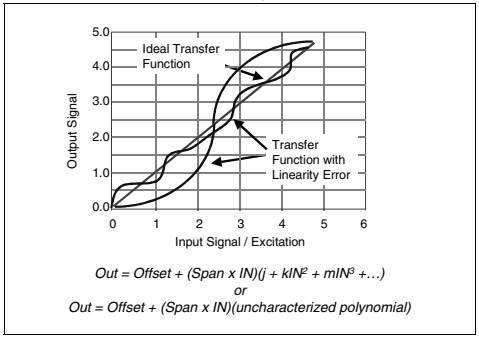


FIGURE 7: *The linearity error of a sensor or system can sometimes be modeled and understood, which allows the designer to use predetermined algorithms in the MCU to minimize their affect. However, typically, these errors are not easily predicted and difficult to calibrate out of the signal path.*

Linearity errors in this system originate primarily in the sensor and secondarily in the remainder of the signal conditioning circuit.

Firmware Linearization

Linearity errors can be calibrated out of the system in firmware using polynomial calculations if the transfer function is understood or piecewise linearization methods if the transfer function from part to part varies. If piecewise linearization is used, calibration data should be taken from the system and stored in EEPROM. Utilizing the calibration data, piecewise linearization is easily implemented using two 16 bit unsigned subtractions, one 16 bit unsigned multiplication and one 16 bit unsigned divide.

$$X_{CAL} = X_{FULL} / (Y_{FULL} - Y_{OFF}) \times (Y_{SAMP} - Y_{ZERO})$$

where:

X_{CAL} is equal to the calibrated results

X_{FULL} is the ideal full scale response saved in EEPROM

Y_{FULL} is the measured full scale response saved in EEPROM

Y_{OFF} is the measured offset error saved in EEPROM

Y_{SAMP} is the measured sample that requires linearization and calibration

This style of linearization correction also removes offset and span errors from the digital results.

Hardware Linearization

There are two components that generate linearity errors. The sensor can contribute up to ±0.25%FS error. The capacitor in this circuit can also contribute an appreciable error if care is not taken in limiting the charge and discharge range of the device. If the R/C time constant of the circuit is greater than the inverse of the sample frequency, the non-linearity of this time response will cause a linearity error in the system.

In this case the R/C time constant is equal to:

$t_{RC} = R_1 // R_2 \times C_{INT}$

$t_{RC} = 2.67k\Omega // 165k\Omega \times 1\mu F$

$t_{RC} = 2.627msec$

also, $t_{RC} <= t_{SAMPLE} / 6.5$

The maximum voltage deviation due to the non-linearity of the R/C network is ~10mV. This is below a 0.2% error. If a lower sampling frequency is used, the integrating capacitor must be increased in value.

SYSTEM NOISE ERRORS

Noise can plague the best of circuits, particularly circuits that have large analog segments. An effective way to approach noise problems is to use a basic list of guidelines in conjunction with a working knowledge of noise fundamentals. The checklist that every designer should have on hand includes:

1. Are bypass capacitors included in the design?
2. Is a low impedance ground plane implemented to minimize any ground noise across sensitive analog parts?
3. Are appropriate anti-aliasing filters used in front of the A/D converter?
4. Are the devices in the circuit too noisy?

Firmware Noise Reduction

The A/D converter described in this application note was modeled after a classic first order delta-sigma topology. In the digital domain, the data collection algorithm implements a simple average engine by default. This style of averaging, otherwise known as digital filtering, is also called a single order sinc filter or Finite Impulse Response (FIR) filter.

Further noise reduction algorithms can be implemented with the PIC MCU which will produce a system with higher accuracy. As an example, a third order FIR filter can be implemented with the following calculations in code:

$y(n) = (2x(n) + x(n-1) + x(n-2) + x(n-3) + x(n-4) + x(n-5) + x(n-6)) / 8$

where:

n identifies the measurement sample

y(n) is the output digital results

x(n) is the measurement sample results

This filter is also known as a $sinc^3$ filter.

Other filter types, such as the Infinite Impulse Response (IIR) filter or a decimation filter can also be used to improve accuracy. A detailed discussion of these filters are beyond the scope of this application note, however, references have been provided for further reading.

Hardware Noise Reduction

In Figure 2, the R/C network that is used to implement the integrator function also serves as a low pass filter. This low pass filter is equal to:

$f_{3dB} = 1 / (2 \pi R_1 // R_2 \times C_{INT})$

Further noise reduction can be implement by adding a second modulator stage at the input so this system. This implementation is shown in Figure 8.

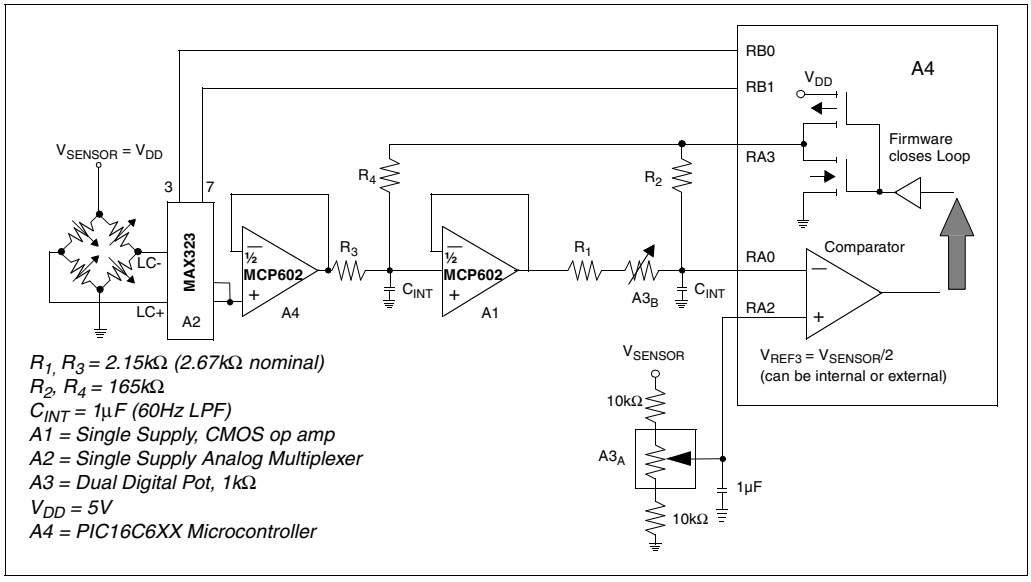


FIGURE 8: *An additional modulator stages can reduce noise in the system even further. In this circuit, one modulator stage is added which improves the system from a 9.9-bit accurate (rms) system to an 11.1-bit accurate system.*

REFERENCES

Peter, Baker, Butler, Darmawaskita, "Making a Delta-Sigma Converter Using A Microcontroller's Analog Comparator Module", *AN700, Microchip Technology, Inc.*

Baker, Bonnie C., "Anti-aliasing, Analog Filters for Data Acquisition Systems", *AN699, Microchip Technology, Inc.*

Baker, Bonnie C., "Layout Tips for 12-Bit Converter Applications", *AN688, Microchip Technology, Inc.*

Morrison, Ralph, "Noise and Other Interfering Signals", *John Wiley & Sons, Inc., 1192*

Baker, Bonnie C., "Analog Circuit Noise Sources and Remedies", *EDTN Internet Magazine, Analog Avenue Tech Notes, Oct. 1998*

Baker, Bonnie C., "Noise Sources in Applications Using Capacitive Coupled Isolated Products", *AB-047, Burr-Brown Corp.*

Palacheria, Amar, "Implementing IIR Digital Filters", *AN540, Microchip Technology, Inc.*

Norsworthy, Schreier, Temes, "Delta-Sigma A/D Converters: Theory, Design, and Simulation", *IEEE Press*

NOTES:

AN719

Interfacing Microchip's MCP3201 Analog-to-Digital Converter to the PICmicro® Microcontroller

Author: *Richard L. Fischer*
Microchip Technology Inc.

INTRODUCTION

Many of the embedded control systems designed today require some flavor of a Analog-to-Digital (A/D) Converter. Embedded system applications such as data acquisition, sensor monitoring and instrumentation and Control all have varying A/D Converter requirements.

For the most part, these A/D Converter requirements are a combination of performance, cost, package size, and availability. Microchip offers a variety of solutions to meet these design requirements. The first possible solution is to implement the PICmicro® microcontroller (MCU). The PICmicro MCU offers many options for smart solutions. One of these features is the A/D Converter module. These A/D Converter modules are primarily successive approximation register (SAR) type and range in functionality from 8- to 12-bit with channel size ranges of 4 to 16. For example, the PIC16C77 has 8-channels of 8-bit A/D Converter, while the PIC17C766 has 16-channels of 10-bit A/D Converter.

These on-board A/D Converter modules fit well into embedded applications, which requires a 10-35ksps A/D Converter.

For those applications which require a higher performance or remote sense capability, the Microchip MCP3201, 12-bit A/D Converter fits very nicely.

The MCP3201 employs a classic SAR architecture. The device uses an internal sample and hold capacitor to store the analog input while the conversion is taking place. Conversion rates of 100ksps are possible on the MCP3201. Minimum clock speed (10kHz or 625sps, assuming 16 clocks) is a function of the capacitors used for the sample and hold.

The MCP3201 has a single pseudo-differential input. The (IN–) input is limited to ±100mV. This can be used to cancel small noise signals present on both the (IN+) and (IN–) inputs. This provides a means of rejecting noise when the (IN–) input is used to sense a remote signal ground. The (IN+) input can range from the (IN–) input to V_{REF}.

The reference voltage for the MCP3201 is applied to V_{REF} pin. V_{REF} determines the analog input voltage range and the LSB size, i.e.:

$$LSB\ size = \frac{V_{REF}}{2^{12}}$$

As the reference input is reduced, the LSB size is reduced accordingly.

Communication with the MCP3201 is accomplished using a standard SPI™ compatible serial interface. This interface allows direct connection to the serial ports of MCUs and digital signal processors.

In order to simplify the design process for implementing the MCP3201, Microchip has written C and assembly code routines for a PIC16C67 to communicate with the MCP3201 A/D Converter.

Figure 1 shows the hardware schematic implemented in this application. Appendix A contains a listing of the C source code. Appendix B contains a listing of the assembly source code.

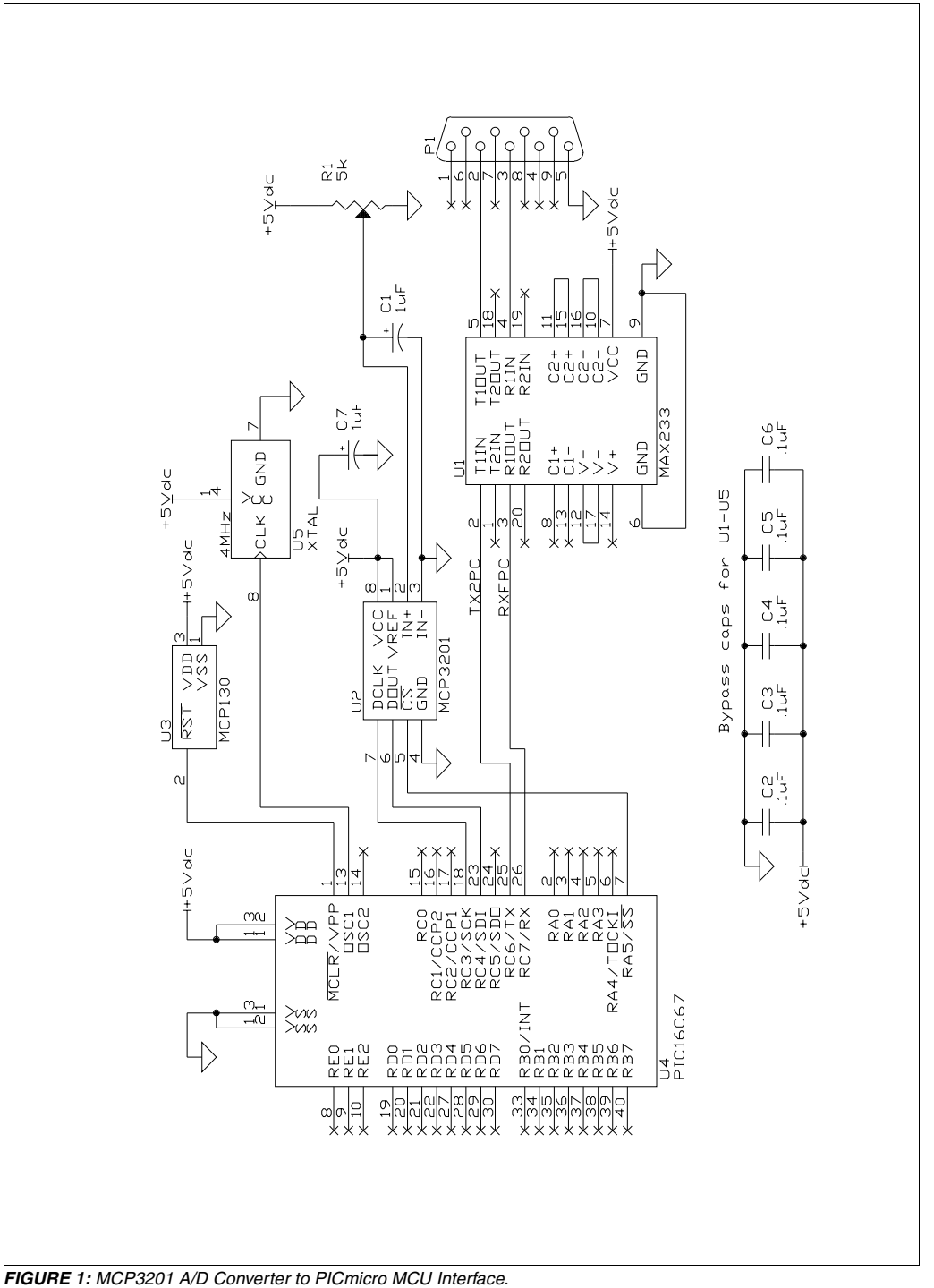


FIGURE 1: *MCP3201 A/D Converter to PICmicro MCU Interface.*

CIRCUIT DESCRIPTION

The serial interface of the Microchip MCP3201 A/D Converter has three wires, a serial clock input (DCLK), the serial data output (D_{OUT}) and the chip select input signal ($\overline{CS}$/SHDN). For this simple circuit interface, the PICmicro PIC16C67 SPI port is used. PortC:<3> is configured for the serial clock and PortC:<4> is the data input to the PICmicro. The SPI clock rate for this application is set at 1MHz.

The PIC16C67 is configured in the master mode with its CKP bit set to logic 1 and CKE bit set to logic 0. This configuration is the SPI bus mode 1,1.

A conversion is initiated with the high to low transition of $\overline{CS}$/SHDN (active low). The chip select is generated by PORTA:<5> of the PICmicro. The device will sample the analog input from the rising edge on the first clock after $\overline{CS}$ goes low for 1.5 clock cycles. On the falling edge of the second clock, the device will output a low null bit. The next 12 clocks will output the result of the conversion with the MSB first (See Figure 2 and Figure 3). Data is always output from the device on the falling edge of the clock. If the device continues to receive clocks while $\overline{CS}$/SHDN is low, the device will output the conversion LSB first. If more clocks are provided to the device while $\overline{CS}$/SHDN is still low (after the LSB first data has been transmitted), the device will clock out zeros indefinitely.

As the analog input signal is applied to the IN+ and IN- inputs, it is ratioed to the V_{REF} input for conversion scaling.

$$\textit{Digital output code} = \frac{V_{IN} \times F.S.}{V_{REF}}$$

Where:

V_{IN} = analog input voltage V(IN+) - V(IN–)

V_{REF} = reference voltage

F.S. = full scale = 4096

V_{REF} can be sourced directly from V_{DD} or can be supplied by an external reference. In either configuration, the V_{REF} source must be evaluated for noise contributions during the conversion. The voltage reference input, V_{REF} of the MCP3201 ranges from 250mV to $5V_{DC}$ which approximately translates to a corresponding LSB size from 61µV to 1.22mV per bit.

$$1.22mV = \frac{5V_{DC}}{2^{12}\ bits}$$

For this simple application, the MCP3201 voltage reference input is tied to $5V_{DC}$. This translates to a 1.22mV / bit resolution for the A/D Converter module. The voltage input to the MCP3201 is implemented with a multi-turn potentiometer. The output voltage range of this passive driver is approximately $0V_{DC}$ to $5V_{DC}$.

Finally, a simple RS-232 interface is implemented using the USART peripheral of the PICmicro and a MAX233 transceiver IC. The USART transmits the captured A/D Converter binary value, both in ASCII and corresponding voltage to the PC terminal at 9600 baud.

With a few discrete components, a MCP3201 A/D Converter IC., and a PICDEM-2 demonstration board, this simple application can be implemented.

As with all applications which require moderate to high performance A/D Converter operation, proper grounding and layout techniques are essential in achieving optimal performance. Proper power supply decoupling and input signal and V_{REF} parameters must be considered for noise contributions.

SOURCE CODE DESCRIPTION

The code written for this application performs six functions:

1. PICmicro Initialization
2. A/D Conversion
3. Conversion to ASCII
4. Conversion to Decimal
5. Conversion to Voltage (*C code only)
6. Transmit ASCII, Decimal and Voltage to PC for display.

C CODE:

Upon power up, three initialization routines are called and executed. These routines initialize the PICmicro Port pins, USART peripheral and SSP module for SPI functionality. The default PICmicro SPI bus mode is 1,1. To place the PICmicro in SPI bus mode 0,0, comment out the “#define mode11” definition statement and rebuild the project.

Upon completion of the initialization routines, the main code loop is entered and executed every ~150ms. This continuous loop consists of performing an analog conversion, transmitting the results to the PC for display, delaying for ~150ms and then repeating the loop.

The A/D conversion sequence is initiated every time $\overline{CS}$/SHDN is asserted. PortA:<5> is used as the $\overline{CS}$/SHDN to the MCP3201. After asserting PortA:<5>, the SSPBUF register is written to, for initiating a SPI bus cycle. When the SPI cycle is complete, (BF flag is set to logic 1), the received data is read from the SSPBUF register and written to the RAM array variable "adc_databyte[1]". The SSPBUF register is again written to, which initiates a SPI bus cycle, and the second 8-bits are received and written to the RAM array variable "adc_databyte[0]". The $\overline{CS}$/SHDN is then negated and the MCP3201 enters into the shutdown mode.

Next, the “Display_Adc_Result” routine is called and executed. Here the composite result, located in array variable “adc_databyte” is right adjusted one bit location. Then a printf statement is executed which formats

and sends the data to the USART for transmission to the PC for display. The data output is in three formats: ASCII, Decimal and Voltage.

ASSEMBLY CODE:

Upon power up, three initialization routines are called and executed. These routines initialize the PICmicro Port pins, USART peripheral and SSP module for SPI functionality. The default PICmicro SPI bus mode is 1,1. To place the PICmicro in SPI bus mode 0,0, comment out the "#define mode11" statement and rebuild the project.

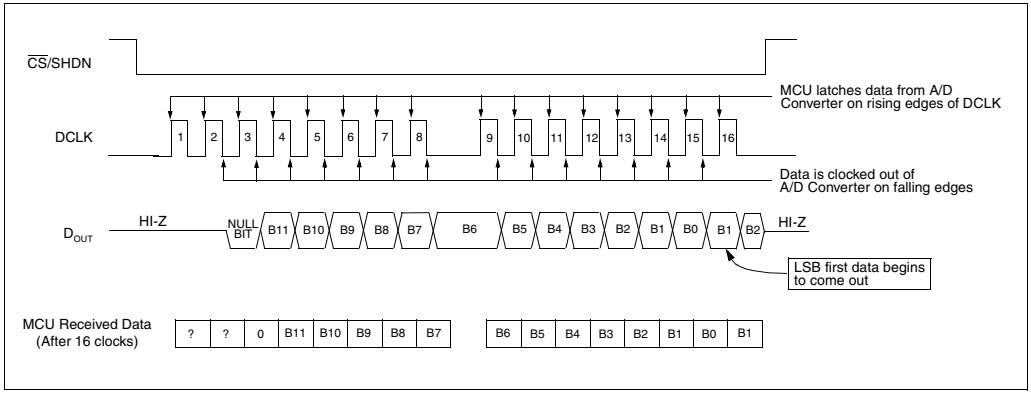


FIGURE 2: *SPI Communication using 8-bit segments (Mode 0,0: DCLK idles low).*

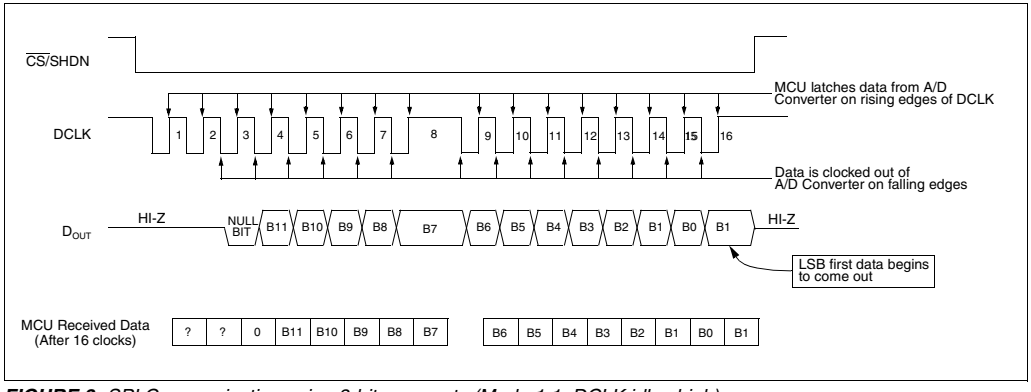


FIGURE 3: *SPI Communication using 8-bit segments (Mode 1,1: DCLK idles high).*

Upon completion of the initialization routines, the main code loop is entered and executed every ~150ms. This continuous loop consists of performing an analog conversion, converting the A/D Converter binary data into Decimal and ASCII and then transmitting the results to the PC for display, delaying for ~150ms and then repeating the loop.

The A/D conversion sequence is initiated every time $\overline{CS}$/SHDN is asserted. PortA:<5> is used as the $\overline{CS}$/SHDN to the MCP3201. After asserting PortA:<5>, the SSPBUF register is written to, for initiating a SPI bus cycle. When the SPI cycle is complete, (BF flag is set to logic 1), the received data is read from the SSPBUF register and written to the RAM variable "adc_result+1". The SSPBUF register is again written to, which initiates a SPI bus cycle, and the second 8-bits are received and written to the RAM variable "adc_result". Here the composite result, located in variable adc_result is right adjusted one bit location. The $\overline{CS}$/SHDN is negated and the MCP3201 enters into the shutdown mode.

Next, the “Hex_Dec” and “Hex_Ascii” routines are executed which convert the raw A/D Converter binary data into Decimal and ASCII values. Then, the ”Display_Data” routine is executed which sends the data to the USART for transmission to the PC for display.

REFERENCES

Williams, Jim, “Analog Circuit Design”, *Butterworth-Heinemann.*

Baker, Bonnie, “Layout Tips for 12-bit A/D Converter Applications”, *AN688, Microchip Technology Inc.*

MCP3201 12-bit A/D Converter with SPI Serial Interface, *Microchip Technology, Document # DS21290B, 1999.*

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX A:

```
/*****************************************************************************
*                                                                            *
*   Interfacing Microchip's MCP3201 ADC to the PICmicro MCU                  *
*                                                                            *
*****************************************************************************
*                                                                            *
*   Filename:       mcp3201.c                                                *
*   Date:           06/30/99                                                 *
*   File Version:   1.00                                                     *
*                                                                            *
*   Compiler:       Hi-Tech PIC C Compiler V7.84 PL1                         *
*                   MPLAB V4.12.00                                           *
*                                                                            *
*   Author:         Richard L. Fischer                                       *
*                   Microchip Technology Incorporated                        *
*                                                                            *
*****************************************************************************
*                                                                            *
*   Files required:                                                          *
*                                                                            *
*                   pic.h       - Hi-Tech provided file                      *
*                   stdio.h     - Hi-Tech provided file                      *
*                   cnfig67.h                                                *
*                   mcp3201.h                                                *
*                                                                            *
*                   mcp3201.c                                                *
*                   mprnt.c     - Hi-Tech provided file                      *
*                                                                            *
*****************************************************************************
*                                                                            *
*                                                                            *
*  This code demonstrates how the Microchip MCP3201 Analog-to-Digital*
*  Converter (ADC) is interfaced to the Synchronous Serial Peripheral*
*  (SSP) of the PICmicro MCU. For this application note the PICmicro *
*  PIC16C67 is selected. The interface uses two Serial Peripheral    *
*  Interface (SPI) lines (SCK, SDI) on the PICmicro for the clock    *
*  (SCK) and data in (SDI). A chip select (CS) to the MCP3201 is     *
*  generated with a general purpose port line PORTA:<5>. The simple  *
*  application uses Mode 1,1 to define bus clock polarity and        *
*  phase.                                                            *
*                                                                    *
*  For this application, the SPI data rate is set to one fourth      *
*  (FOSC/4) of the microcontroller clock frequency. The PIC16C67     *
*  device clock frequency used for this application is 4MHz. This    *
*  translates to an ADC throughput of approximately 62.5kHz. In      *
*  order to obtain the maximum throughput (100kHz) from the          *
*  MCP3201 ADC the PIC16C67 should be clocked at 6.4Mhz.             *
*                                                                    *
*                                                                    *
*                                                                    *
*****************************************************************************/

    #include     <pic.h>                          // processor if/def file
    #include     <stdio.h>
    #include     "cnfig67.h"                      // configuration word definitions
    #include     "mcp3201.h"

_CONFIG    ( CONBLANK & BODEN_ON & PWRTE_ON & CP_OFF & WDT_OFF & XT_OSC );

/* SPI Bus mode selection */
#define mode11                                   // comment out and rebuild for mode 00
```

```
/*********************************************************************

  MAIN PROGRAM BEGINS HERE

*********************************************************************/

void main( void )
{
    Init_Ports();                                   // initialize ports
    Init_SSP();                                     // initialize SSP module
    Init_Usart();                                   // initialize USART module

    while ( TRUE )                                  // loop forever
    {
         Read_Adc( );                               // initiate MCP3201 conversion and read result
         Display_Adc_Result();                      // display results via USART to PC
         Delay_10mS( 15 );                          // 150mS delay
    }
}



void Delay_10mS( char loop_count )                  // approximate 10mS base delay
{
    unsigned int inner;                             // declare integer auto variable
    char outer;                                     // declare char auto variable

    while ( loop_count )                            // stay in loop until done
    {
       for ( outer = 9; outer > 0; outer-- )
        for ( inner = 249; inner > 0; inner-- );
       loop_count--;
    }
}



void putch( char data )
{
    while ( !TRMT );                                // wait until TSR is empty
    TXREG = data;                                   // write data to USART
}



void Read_Adc( void )
{
    CS = 0;                                         // assert MCP3201 chip select
    SSPBUF = 0x01;                                  // initiate a SPI bus cycle
    while ( !STAT_BF );                             // wait until cycle completes
    adc.databyte[1] = SSPBUF;                       // transfer ADC MSbyte into buffer

    SSPBUF = 0x81;                                  // initiate a SPI bus cycle
    while ( !STAT_BF );                             // wait until cycle completes
    CS = 1;                                         // negate MCP3201 chip select
    adc.databyte[0] = SSPBUF;                       // transfer ADC LSbyte into buffer
}



void Display_Adc_Result( void )
{
    double temp;                                    // define auto type variable
    adc.result >>= 1;                               // adjust composite integer for 12 valid bits
    adc.result &= 0x0FFF;                           // mask out upper nibble of integer
    temp = ( adc.result * 0.001225585 );            // compute floating point result
```

```
    printf( "Hex->0x%X  :  Decimal->%u  :  %4.3f Vdc\n\r", adc.result, adc.result, temp );
}



void Init_Usart( void )
{
    SPBRG = 25;                                 // set baud rate for 9600 @ 4MHz
    TXSTA = 0x24;                               // BRGH = 1, enable transmitter
    RCSTA = 0x90;                               // enable serial port
}



void Init_SSP( void )
{
#ifdef mode11
    SSPSTAT = 0b00000000;                       // Master sample data in middle, data xmt on
                                                // rising edge
    SSPCON = 0b00110000;                        // enable Master SPI, bus mode 1,1, FOSC/4

#else if
    SSPSTAT = 0b01000000;                       // Master sample data in middle, data xmt on
                                                // rising edge
    SSPCON = 0b00100000;                        // enable Master SPI, bus mode 0,0, FOSC/4

#endif
}



void Init_Ports( void )
{
    PORTA = 0b100000;                           // set PORTA data latches to initial state
    PORTB = 0x00;                               // set PORTB data latches to initial state
    PORTC = 0b11010000;                         // set PORTC data latches to initial state
    PORTD = 0x00;                               // set PORTD data latches to initial state
    PORTE = 0x00;                               // set PORTE data latches to initial state

    TRISA = 0b000000;                           // set PORTA pin direction
    TRISB = 0x00                                // set PORTB pin direction
    TRISC = 0b11010000;                         // set PORTC pin direction
    TRISD = 0x00;                               // set PORTD pin direction
    TRISE = 0x00;                               // set PORTE pin direction
}
```

```
/************************************************************************
*                                                                       *
*   Filename:       mcp3201.h                                           *
*   Date:           06/30/99                                            *
*   File Version:   1.00                                                *
*                                                                       *
*                                                                       *
************************************************************************/



// FUNCTION PROTOTYPES DECLARED HERE

void Read_Adc( void );
void Display_Adc_Result( void );
void Delay_10mS( char loop_count );
void Init_Usart( void );
void Init_SSP( void );
void Init_Ports( void );


union {
           char databyte[2];                    // declare temp array for adc data
           unsigned int result;                 // declare integer for adc result
} adc;                                          // define union variable


#define TRUE    1

#define PortBit(port,bit)    ((unsigned)&(port)*8+(bit))

static bit CS  @  PortBit(PORTA,5);             // MCP3201 Chip Select
```

```
/*********************************************************************
*                                                                    *
*   Filename:       cnfig67.h                                        *
*   Date:           06/30/99                                         *
*   File Version:   1.00                                             *
*                                                                    *
*                                                                    *
*********************************************************************/

/*****  CONFIGURATION BIT DEFINITIONS FOR PIC16C67 PICmicro  *****/

#define        CONBLANK          0x3FFF

#define        CP_ALL            0x00CF
#define        CP_75             0x15DF
#define        CP_50             0x2AEF
#define        CP_OFF            0x3FFF
#define        BODEN_ON          0x3FFF
#define        BODEN_OFF         0x3FBF
#define        PWRTE_OFF         0x3FFF
#define        PWRTE_ON          0x3FF7
#define        WDT_ON            0x3FFF
#define        WDT_OFF           0x3FFB
#define        LP_OSC            0x3FFC
#define        XT_OSC            0x3FFD
#define        HS_OSC            0x3FFE
#define        RC_OSC            0x3FFF
```

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX B:

```
;*****************************************************************************
;                                                                            *
;   Interfacing Microchip's MCP3201 ADC to the PICmicro MCU                  *
;                                                                            *
;*****************************************************************************
;                                                                            *
;   Filename:       mcp3201.asm                                              *
;   Date:           06/30/99                                                 *
;   File Version:   1.00                                                     *
;                                                                            *
;   Assembler:      MPASM  V2.30.00                                          *
;   Linker:         MPLINK V1.30.01                                          *
;                   MPLAB  V4.12.00                                          *
;                                                                            *
;   Author:         Richard L. Fischer                                       *
;   Company:        Microchip Technology Incorporated                        *
;                                                                            *
;*****************************************************************************
;                                                                            *
;   Files required:                                                          *
;                                                                            *
;                   mcp3201.asm                                              *
;                   hexdec.asm                                               *
;                   hexascii.asm                                             *
;                                                                            *
;                   p16c67.inc                                               *
;                   16c67.lkr                                                *
;                                                                            *
;                                                                            *
;*****************************************************************************
;                                                                            *
;  This code demonstrates how the Microchip MCP3201 Analog-to-Digital*
;  Converter (ADC) is interfaced to the Synchronous Serial Peripheral*
;  (SSP) of the PICmicro MCU. For this application note the PICmicro *
;  PIC16C67 is selected. The interface uses two Serial Peripheral    *
;  Interface (SPI) lines (SCK, SDI) on the PICmicro for the clock    *
;  (SCK) and data in (SDI). A chip select (CS) to the MCP3201 is     *
;  generated with a general purpose port line PORTA:<5>. The simple  *
;  application uses Mode 1,1 to define bus clock polarity and        *
;  phase.                                                            *
;                                                                    *
;  For this application, the SPI data rate is set to one fourth      *
;  (FOSC/4) of the microcontroller clock frequency. The PIC16C67     *
;  device clock frequency used for this application is 4MHz. This    *
;  translates to an ADC throughput of approximately 62.5kHz. In      *
;  order to obtain the maximum throughput (100kHz) from the          *
;  MCP3201 ADC the PIC16C67 should be clocked at 6.4Mhz.             *
;                                                                    *
;                                                                    *
;                                                                    *
;*****************************************************************************/


    list         p=16c67                          ; list directive to define processor
    #include     <p16c67.inc>                     ; processor specific variable definitions

    __CONFIG   _BODEN_ON & _PWRTE_ON & _CP_OFF & _WDT_OFF & _XT_OSC


#define mode11                                    ; if SPI bus mode 1,1 is desired
                                                  ; else comment out and rebuild for mode 0,0
```

```
;***** VARIABLE DEFINITIONS

TEMP_VAR        UDATA       0x20                ;
adc_result      RES         2                   ; variable used for context saving
offset          RES         1
temp            RES         1


TEMP_VAR1       UDATA_OVR                       ; create udata overlay section
counthi         RES         1
countlo         RES         1


            GLOBAL   adc_result                 ; make variables available to other modules
            EXTERN   Hex_Dec                    ; reference linkage
            EXTERN   Hex_Ascii                  ; reference linkage
            EXTERN   adc_temph, adc_templ       ; reference linkage
            EXTERN   thous                      ; reference linkage


#define       CS      PORTA,5                   ; MCP3201 Chip Select
#define       CR      0x0D                      ; macro for carriage return
#define       LF      0x0A                      ; macro for line feed


;*****************************************************************

RESET_VECTOR    CODE        0x000               ; processor reset vector
            movlw    high start                 ; move literal into W
            movwf    PCLATH                     ; initialize PCLATH
            goto     start                      ; go to beginning of program


INT_VECTOR      CODE        0x004               ; interrupt vector location
; no interrupt code needed for this application



MAIN            CODE        0x040               ; set code section to start at 0x040
start
            call        Init_Ports              ; initialize ports
            call        Init_SSP                ; initialize SSP module
            call        Init_Usart              ; initialize USART module

forever     call        Read_Adc                ; read MCP3201 ADC
            call        Hex_Dec                 ; convert adc_result to decimal
            call        Hex_Ascii               ; convert adc_result to ASCII
            call        Display_Data            ; display data to PC
            call        Delay_150mS             ; 150mS delay
            goto        forever                 ; continuos loop



; Read MCP3201 ADC for 2 bytes
Read_Adc
            banksel     PORTA                   ; linker to select SFR bank
            bcf         CS                      ; assert MCP3201 chip select
            movlw       0x01                    ; move literal into W
            banksel     SSPBUF                  ; linker to select SFR bank
            movwf       SSPBUF                  ; initiate SPI bus cycle
            banksel     SSPSTAT                 ; linker to select SFR bank
spi_busy1   btfss       SSPSTAT,BF              ; test, is bus cycle complete?
            goto        spi_busy1               ; wait, bus cycle not complete
            banksel     SSPBUF                  ; linker to select SFR bank
            movf        SSPBUF,w                ; read SSPBUF and place into W
            banksel     adc_result              ; linker to select GPR bank
```

```
            movwf       adc_result+1                ; write SSPBUF to adc_result

            movlw       0x81                        ; move literal into W
            banksel     SSPBUF                      ; linker to select SFR bank
            movwf       SSPBUF                      ; initiate SPI bus cycle
            banksel     SSPSTAT                     ; linker to select SFR bank
spi_busy2   btfss       SSPSTAT,BF                  ; test, is bus cycle complete?
            goto        spi_busy2                   ; wait, bus cycle not complete
            banksel     PORTA                       ; linker to select SFR bank
            bsf         CS                          ; negate MCP3201 chip select
            movf        SSPBUF,w                    ; read SSPBUF and place into W
            banksel     adc_result                  ; linker to select GPR bank
            movwf       adc_result                  ; write SSPBUF to adc_result

            rrf         adc_result+1,f              ; adjust MSB 1 position right
            rrf         adc_result,f                ; adjust LSB 1 position right and include carry
            movlw       0x0F                        ; move literal into W
            andwf       adc_result+1,f              ; mask out upper nibble of ADC result

            movf        adc_result,w                ; move adc_result LSB into W
            movwf       adc_templ                   ; save W into temp register
            movf        adc_result+1,w              ; move adc_result MSB into W
            movwf       adc_temph                   ; save W into temp register
            return                                  ; return from subroutine



; Display ADC data ( ASCII and DECIMAL ) to USART
Display_Data
            banksel     offset                      ; linker to select GPR bank
            clrf        offset                      ; initialize table index value
            movlw       high  msg1                  ; move high byte of table address -> W
            movwf       PCLATH                      ; initialize PCLATH

txlp1       movf        offset,w                    ; move offset value into W
            call        msg1                        ; retrieve table element
            movwf       temp                        ; move element into temp
            btfsc       temp,7                      ; test for end of string
            goto        send_hex                    ; end of message so send the data
            banksel     TXREG                       ; linker to select SFR bank
            movwf       TXREG                       ; initiate USART transmission
            banksel     TXSTA                       ; linker to select SFR bank
            btfss       TXSTA,TRMT                  ; test if TSR is empty
            goto        $-1                         ; stay in testing loop
            banksel     offset                      ; linker to select GPR bank
            incf        offset,f                    ; increment table index
            goto        txlp1                       ; stay in transmit loop

send_hex    movlw       adc_temph                   ; obtain variable address
            movwf       FSR                         ; initialize FSR as pointer
send_hex1   movf        INDF,w                      ; retrieve data byte
            banksel     TXREG                       ; linker to select SFR bank
            movwf       TXREG                       ; initiate USART transmission
            banksel     TXSTA                       ; linker to select bank
            btfss       TXSTA,TRMT                  ; test if TSR is empty
            goto        $-1                         ; stay in testing loop
            incf        FSR,f                       ; update pointer
            movlw       adc_temph+4                 ; compose end of string address value
            subwf       FSR,w                       ; do compare
            btfss       STATUS,C                    ; done with sending data
            goto        send_hex1                   ; no, so send some more


            banksel     offset                      ; linker to select GPR bank
            clrf        offset                      ; initialize table index value
txlp2       movf        offset,w                    ; move offset value into W
```

```
            call        msg2                    ; retrieve table element
            movwf       temp                    ; move element into temp
            btfsc       temp,7                  ; test for end of string
            goto        send_dec                ; end of message so send the data
            banksel     TXREG                   ; linker to select SFR bank
            movwf       TXREG                   ; initiate USART transmission
            banksel     TXSTA                   ; linker to select SFR bank
            btfss       TXSTA,TRMT              ; test if TSR is empty
            goto        $-1                     ; stay in testing loop
            banksel     offset                  ; linker to select GPR bank
            incf        offset,f                ; increment table index
            goto        txlp2                   ; stay in transmit loop


send_dec    movlw       thous                   ; obtain variable address
            movwf       FSR                     ; initialize FSR as pointer
send_dec1   movf        INDF,w                  ; retrieve data byte
            banksel     TXREG                   ; linker to select SFR bank
            movwf       TXREG                   ; initiate USART transmission
            banksel     TXSTA                   ; linker to select SFR bank
            btfss       TXSTA,TRMT              ; test if TSR is empty
            goto        $-1                     ; stay in loop
            incf        FSR,f                   ; update pointer
            movlw       thous+4                 ; compose end of string address value
            subwf       FSR,w                   ; do compare
            btfss       STATUS,C                ; done with sending data
            goto        send_dec1               ; no, so send some more

            movlw       CR                      ; move literal into W
            banksel     TXREG                   ; linker to select SFR bank
            movwf       TXREG                   ; initiate USART transmission
            banksel     TXSTA                   ; linker to select SFR bank
            btfss       TXSTA,TRMT              ; test if TSR is empty
            goto        $-1                     ; no, so stay in loop

            movlw       LF                      ; move literal into W
            banksel     TXREG                   ; linker to select SFR bank
            movwf       TXREG                   ; initiate USART transmission
            banksel     TXSTA                   ; linker to select SFR bank
            btfss       TXSTA,TRMT              ; test if TSR is empty
            goto        $-1                     ; no, so stay in loop
            return                              ; return from subroutine



; Delay for ~ 150mS
Delay_150mS
            movlw       D'150'                  ; move literal into W
            banksel     counthi                 ; linker to select GPR bank
            movwf       counthi                 ; initialize upper counter
outer       movlw       D'250'                  ; move literal into W
            movwf       countlo                 ; initialize lower counter
inner       decf        countlo,f               ; decrement counter low
            btfss       STATUS,Z                ; is result == 0
            goto        inner                   ; no, stay in loop
            decf        counthi,f               ; else, decrement count high
            btfss       STATUS,Z                ; is result == 0
            goto        outer                   ; no, so start again
            return                              ; return from subroutine


; Initialize USART Module
Init_Usart  movlw       D'25'                   ; move literal into W
            banksel     SPBRG                   ; linker to select SFR bank
            movwf       SPBRG                   ; set baud rate for 9600 @ 4MHz
            movlw       B'00100100'             ; move literal into W
```

```
          movwf     TXSTA                   ; BRGH = 1, enable transmitter
          movlw     B'10010000'             ; move literal into W
          banksel   RCSTA                   ; linker to select SFR bank
          movwf     RCSTA                   ; enable serial port
          return                            ; return from subroutine


; Initialize SSP Module
Init_SSP
#ifdef mode11
          movlw     B'00110000'             ; move literal into W
          banksel   SSPCON                  ; linker to select SFR bank
          movwf     SSPCON                  ; enable Master SPI, bus mode 1,1, FOSC/4
          banksel   SSPSTAT                 ; linker to select SFR bank
          clrf      SSPSTAT                 ; Master sample data in middle, data xmt on
                                            ; rising edge

#else
          movlw     B'00100000'             ; move literal into W
          banksel   SSPCON                  ; linker to select SFR bank
          movwf     SSPCON                  ; enable Master SPI, bus mode 0,0, FOSC/4
          movlw     B'01000000'             ; move literal into W
          banksel   SSPSTAT                 ; linker to select SFR bank
          movwf     SSPSTAT                 ; Master sample data in middle, data xmt on
                                            ; rising edge
#endif
          return                            ; return from subroutine


; Initialize PORTS
Init_Ports movlw    0x00                    ; move literal into W
          banksel   PORTA                   ; linker to select SFR bank
          movwf     PORTB                   ; set PORTB data latches to initial state
          movwf     PORTD                   ; set PORTD data latches to initial state
          movwf     PORTE                   ; set PORTE data latches to initial state
          movlw     B'100000'               ; move literal into W
          movwf     PORTA                   ; set PORTA data latches to initial state
          movlw     B'11010000              ; move literal into W
          movwf     PORTC                   ; set PORTC data latches to initial state

          banksel   TRISA                   ; linker to select SFR bank
          clrf      TRISA                   ; set PORTA pin direction
          clrf      TRISB                   ; set PORTB pin direction
          clrf      TRISD                   ; set PORTD pin direction
          clrf      TRISE                   ; set PORTE pin direction
          movlw     B'11010000'             ; move literal into W
          movwf     TRISC                   ; set PORTC pin direction
          return                            ; return from subroutine


TABLE_DATA CODE     0x200                   ; table starts here
msg1      addwf     PCL,f                   ; generate computed goto
          DT        "HEX-> 0x",80

msg2      addwf     PCL,f                   ; generate computed goto
          DT        "  :  DECIMAL-> ",80


          END                               ; directive 'end of program'
```

```
;***********************************************************************
;                                                                      *
;   Hex to Decimal conversion of ADC result for display                *
;                                                                      *
;***********************************************************************
;                                                                      *
;   Filename:       hexdec.asm                                         *
;   Date:           06/30/99                                           *
;   File Version:   1.00                                               *
;                                                                      *
;   Assembler:      MPASM  V2.30.00                                    *
;   Linker:         MPLINK V1.30.01                                    *
;                   MPLAB  V4.12.00                                    *
;                                                                      *
;   Author:         Richard L. Fischer                                 *
;   Company:        Microchip Technology Incorporated                  *
;                                                                      *
;***********************************************************************

   #include <p16c67.inc>                          ; processor specific variable definitions


           GLOBAL  Hex_Dec, thous                 ; make subroutine 'Hex_Dec' available to other
                                                    modules
           EXTERN     adc_result                  ; reference linkage

HEXDEC_VAR UDATA       0x30                       ; create udata variable section
thous      RES         1                          ; reserve one location
hunds      RES         1                          ; reserve one location
tens       RES         1                          ; reserve one location
ones       RES         1                          ; reserve one location


           GLOBAL  thous, hunds, tens, ones


; *****    Subroutine begins here

HEXDEC     CODE                                   ; create code section "HEXDEC"
Hex_Dec
           banksel    thous                       ; linker to select GPR bank
           clrf       thous                       ; initialize 'thousands' variable
           clrf       hunds                       ; initialize 'hundreds' variable
           clrf       tens                        ; initialize 'tens' variable
           clrf       ones                        ; initialize 'ones' variable

chk_thous  movlw      0x04                        ; move literal into W ... 1024 (0x0400)
           banksel    adc_result+1                ; linker to select GPR bank
           subwf      adc_result+1,w              ; subtract 1024 from adc_result MSB
           btfss      STATUS,C                    ; is adc_result MSB > 1024
           goto       chk_hunds2                  ; no, so check hundreds
           incf       thous,f                     ; else, increment thousands
           movlw      0x04                        ; move literal into W
           subwf      adc_result+1,f              ; subtract 1000 from adc_result MSB
           movlw      D'24'                       ; move literal into W
           addwf      adc_result,f                ; add remainder 24 into adc_result LSB
           btfsc      STATUS,C                    ; was there a carry into adc_result MSB?
           incf       adc_result+1,f              ; yes, so increment
           goto       chk_thous                   ; go check thousands again


chk_hunds2 movlw      0x01                        ; 256 (0x0100)
           subwf      adc_result+1,w              ; subtract 200 from adc_result MSB
           btfss      STATUS,C                    ; is adc_result MSB >= 256
           goto       chk_hunds1                  ; no, so check multiples of 100
           movlw      D'2'                        ; else,
```

```
            addwf       hunds,f                 ; add 2 into hundreds
            movlw       0x01                    ; move literal into W
            subwf       adc_result+1,f          ; subtract 200 from adc_result MSB
            movlw       D'56'                   ; move remainder into W
            addwf       adc_result,f            ; add remainder 56 into adc_result LSB
            btfsc       STATUS,C                ; was there a carry into adc_result MSB
            incf        adc_result+1,f          ; yes, so increment adc_result MSB

            movlw       D'10'                   ; move literal into W
            subwf       hunds,w                 ; check to see if hunds = 1000
            btfss       STATUS,Z                ; is result == 0?
            goto        chk_hunds2              ; no, so check hundreds (200) again
            clrf        hunds                   ; clear hundreds
            incf        thous,f                 ; increment thousands
            goto        chk_hunds2              ; go check hundreds (200) some more


chk_hunds1  movlw       D'100'                  ; move literal into W
            subwf       adc_result,w            ; subtract 100 from adc_result LSB
            btfss       STATUS,C                ; is adc_result >= 100
            goto        chk_tens                ; no so check tens
            incf        hunds,f                 ; else, increment hundreds
            movlw       D'100'                  ; move literal into W
            subwf       adc_result,f            ; reduce hundreds count by 100

            movlw       D'10'                   ; move literal into W
            subwf       hunds,w                 ; check to see if hunds may = 1000
            btfss       STATUS,Z                ; is result == 0?
            goto        chk_hunds1              ; no, so check hundreds (100) again
            clrf        hunds                   ; clear hundreds
            incf        thous,f                 ; increment thousands
            goto        chk_hunds1              ; go check hundreds (100) some more


chk_tens    movlw       D'10'                   ; move literal into W
            subwf       adc_result,w            ; subtract 10 from adc_result LSB
            btfss       STATUS,C                ; is adc_result LSB >= 10
            goto        chk_ones                ; no, so check ones
            incf        tens,f                  ; else, increment tens
            movlw       D'10'                   ; move literal into W
            subwf       adc_result,f            ; reduce tens count by 10
            goto        chk_tens                ; go check tens again


chk_ones    movf        adc_result,w            ; read adc_result LSB and store into W
            movwf       ones                    ; save off as ones
            movlw       0x30                    ; move literal into W
            iorwf       thous,f                 ; compose ASCII byte (thousands)
            iorwf       hunds,f                 ; compose ASCII byte (hundreds)
            iorwf       tens,f                  ; compose ASCII byte (tenths)
            iorwf       ones,f                  ; compose ASCII byte (ones)
            return                              ; return from subroutine

            END                                 ; directive 'end of program'
```

```
;*****************************************************************
;                                                                *
;   Hex to ASCII conversion of ADC result for display            *
;                                                                *
;*****************************************************************
;                                                                *
;   Filename:       hexascii.asm                                 *
;   Date:           06/30/99                                     *
;   File Version:   1.00                                         *
;                                                                *
;   Assembler:      MPASM  V2.30.00                              *
;   Linker:         MPLINK V1.30.01                              *
;                   MPLAB  V4.12.00                              *
;                                                                *
;   Author:         Richard L. Fischer                           *
;   Company:        Microchip Technology Incorporated            *
;                                                                *
;*****************************************************************

  #include <p16c67.inc>                         ; processor specific variable definitions

           GLOBAL     Hex_Ascii                 ; make subroutine 'Hex_Ascii' available to
                                                  other modules
           GLOBAL     adc_temph, adc_templ      ; reference linkage


TEMP_VAR1  UDATA_OVR                            ; create udata overlay section
adc_temph  RES        2
adc_templ  RES        2


HEXASCII   CODE                                 ; create code section "HEXASCII"
Hex_Ascii
           banksel    adc_templ                 ; linker to select GPR bank
           movf       adc_templ,w               ; move copy of adc_result LSB into W
           movwf      adc_templ+1               ; make copy ADC result LSB
           movf       adc_temph,w               ; move copy of adc_result MSB into W
           movwf      adc_temph+1               ; make copy ADC result MSB
           movlw      0x30                      ; move literal into W
           movwf      adc_temph                 ; place a ASCII zero in MS digit location

           swapf      adc_templ,f               ; swap nibbles
           movlw      0x0F                      ; move literal into W
           andwf      adc_templ,f               ; mask out upper nibble
           andwf      adc_templ+1,f             ; mask out upper nibble

           movlw      D'10'                     ; move literal into W
           subwf      adc_templ,w               ; test byte
           btfsc      STATUS,C                  ; was a borrow generated
           goto       add_37L                   ; no, so must be A - F
           movlw      0x30                      ; else it is 0 - 9
           addwf      adc_templ,f               ; compose ASCII byte
chk_lsd    movlw      D'10'                     ; move literal into W
           subwf      adc_templ+1,w             ; test value
           btfsc      STATUS,C                  ; was a borrow generated
           goto       add_37L1                  ; no, so must be A - F
           movlw      0x30                      ; else it is 0 - 9
           addwf      adc_templ+1,f             ; compose ASCII byte

chk_msd    movlw      D'10'                     ; move literal into W
           subwf      adc_temph+1,w             ; test byte
           btfsc      STATUS,C                  ; was a borrow generated
           goto       add_37H                   ; no, so must be A - F
           movlw      0x30                      ; else it is 0 - 9
           addwf      adc_temph+1,f             ; compose ASCII byte
           goto       exit                      ; exit routine
```

```
add_37L     movlw       0x37                    ; move literal into W
            addwf       adc_templ,f             ; compose ASCII character
            goto        chk_lsd                 ; check least significant digit
add_37L1    movlw       0x37                    ; move literal into W
            addwf       adc_templ+1,f           ; compose ASCII character
            goto        chk_msd                 ; check most significant digit

add_37H     movlw       0x37                    ; move literal into W
            addwf       adc_temph+1,f           ; compose ASCII character

exit        return                              ; return from subroutine

            END                                 ; directive 'end of program'
```

NOTES:

AN722

Operational Amplifier Topologies and DC Specifications

Author: Bonnie C. Baker
Microchip Technology Inc.

INTRODUCTION

Operational amplifiers (op amps) are as prolific in analog circuits as salt and pepper is on food. They are sprinkled throughout the sensor data acquisition system, performing a variety of functions. For instance, at the sensor interface, amplifiers are used to buffer and gain the sensor output. The current or voltage excitation to the sensor, quite often is generated by an amplifier circuit. Following the front end sensor circuitry, an op amp is used to implement a low pass, band pass or high pass filter. In this portion of the circuit, gain stages are also implemented using programmable gain amplifiers or instrumentation amplifiers whose building blocks are the op amp. Analog-to-Digital (A/D) converters are most typically driven by an amplifier in order to achieve good converter performance.

Each one of these amplifier applications place unique demands on the device, so that one performance specification may be critical in one circuit, but not necessarily in another. This application note defines the DC specifications of op amps and presents circuit applications where optimization of a particular specification is critical.

DEFINING THE OP AMP

Ideal Specifications

The op amp can be simply defined as an analog gain block with two signal inputs, two power supply connections and one output, as shown in Figure 1.

The input stage of the op amp has two terminals, the non-inverting (V_{IN+}) and inverting (V_{IN-}) inputs. For the ideal voltage feedback amplifier, both inputs are matched having no leakage current, infinite input impedance, infinite common mode rejection, zero noise and zero offset voltage (V_{OS}) between the terminals.

The power supply terminals (V_{DD} and V_{SS}) of the ideal op amp, have no minimum or maximum voltage restrictions. Additionally, the current from the power supply through the amplifier (I_{SUPPLY}, I_{DD} or I_Q) is zero and any variation in the power supply voltage does not introduce errors into the analog signal path.

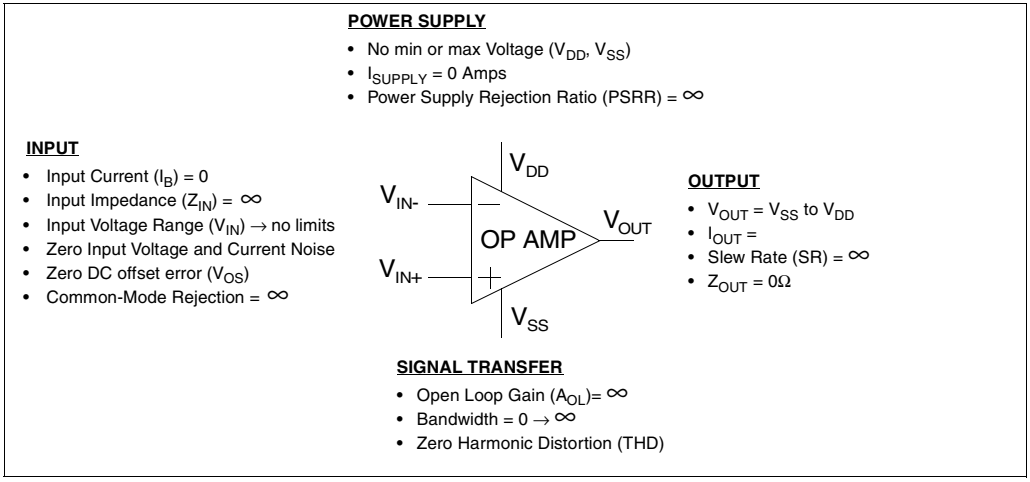


FIGURE 1: *The ideal op amp description can be separated into four basic categories: input, power supply, output, and signal transfer.*

In terms of the amplifier output, the swing capability equals or exceeds the voltage restrictions of the power supply. The output current (I_{OUT}) of this terminal can be infinite for indefinite periods of time, without causing reliability or catastrophic failures. The speed (SR) at which the output swings from rail to rail is instantaneous and the output impedance (Z_{OL} or Z_{CL}) is zero.

Finally, the open loop gain of the amplifier block is infinite and the bandwidth of the open loop gain is also infinite. To put the finishing touches on the signal transfer characteristics of the ideal amplifier, signals pass through the device without added distortion (THD) or noise.

Technology Limitations

This ideal amplifier does not exist. Consequently, performance specifications describe the amplifier so that the designer can assess the impact it will have on his circuit.

The errors that appear on the terminals of the op amp are a consequence of the semiconductor process and transistor implementation of the integrated circuit. In terms of the impact of the type of process that is used to design the amplifier, some generalities are summarized in Figure 2. These generalities are just that and not hard and fast rules.

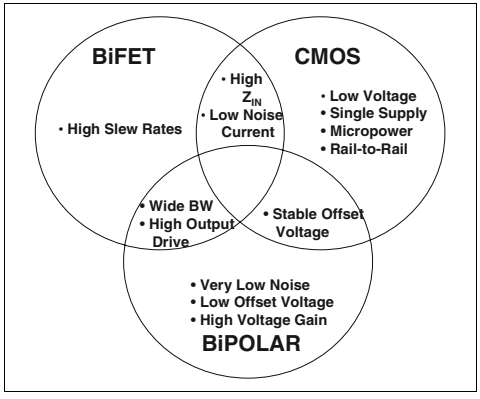


FIGURE 2: *Different IC processes render different advantages for amplifiers. The choices in processes for single supply amplifiers are Bipolar, CMOS and BiFET, which is a combination of Field Effect Transistors (FET) and Bipolar transistors.*

For instance, the BiFET op amp is designed using an FET (Field Effect Transistor) as the device at the input terminals and Bipolar for the remainder of the circuit. Op amp designed with this IC implementation have higher slew rates as compared to the pure Bipolar amplifier and CMOS amplifier.

In contrast, a pure Bipolar amplifier has NPN or PNP transistors at the input terminals. This allows the IC designer to achieve relatively low input offset voltage and voltage noise between the input terminals as well as higher open loop gains.

The commonality between the BiFET and Bipolar amplifiers are that they typically have wider bandwidths and higher output drive capability, as compared to the CMOS amplifier.

CMOS, on the other hand is well known for its low power, single supply op amps. The transistors in this style of amplifier are CMOS, allowing for an infinite input impedance and zero current leakage. This characteristic is similar in BiFET amplifiers. The degradation of these input impedances and leakage currents with the BiFET and CMOS input op amps are due to the required electrostatic discharge (ESD) cells that are added to the input terminals. CMOS amplifiers are also capable of rail-to-rail operation (in analog terms) while still having low quiescent current (current from the power supply).

The op amp specifications can be separated into two general categories, DC and AC. For the remainder of this application note, only the DC specifications will be discussed with accompanying detailed applications where that specification has an impact on the circuit performance. For discussions on AC specifications, refer to the application note from Microchip entitled “Operational Amplifier AC Specifications and Applications”, AN723. (*available December, 1999*)

DC SPECIFICATIONS

The DC specifications discussed in this application note are:

- Input Offset Voltage (V_{OS})
- Input Bias Current (I_B)
- Input Voltage Range (V_{IN} or V_{CM})
- Open Loop Gain (A_{OL})
- Power Supply Rejection (PSRR or PSR)
- Common-mode Rejection (CMRR)
- Output Voltage Swing (V_{OUT}, V_{OH}, or V_{OL})
- Output Resistance (R_{OUT}, R_{OL}, R_{CL}, Z_{OL}, or Z_{CL})
- Power Supply and Temperature Range (V_{SS}, V_{DD}, I_{DD}, and I_Q)

In Figure 3, these parameters are shown in their proper locations to allow for easy circuit evaluation and error analysis.

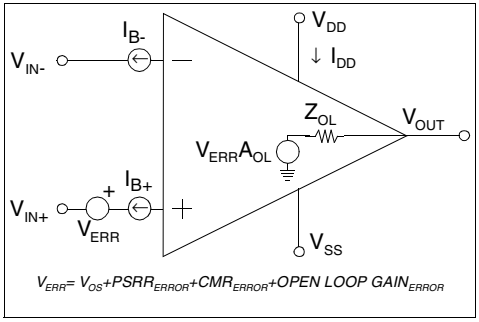


FIGURE 3: *DC parameters for the op amp are modeled in a way to assist definition of specifications and easy error analysis of circuits.*

For the remainder of this application note, these DC specifications will be defined and then evaluated within a sensitive application.

Input Offset Voltage (V_{OS})

Specification Discussion - The input offset voltage specification of an amplifier defines the maximum voltage difference that will occur between the two input terminals in a closed loop circuit while the amplifier is operating in its linear region. The input offset voltage is always specified at room temperature in terms of µV or mV. The over temperature specification can be guaranteed as µV/°C as well as an absolute value of µV or mV. Offset voltage is always modeled as a voltage source at the non-inverting input of the amplifier, as shown in Figure 3.

As with any amplifier specification, offset voltage can vary from part to part and with temperature, as shown in the distribution graphs in the Figure 4. The offset voltage of a particular amplifier does not vary unless the temperature, power supply voltage, common-mode voltage or output voltage changes, as shown in Figure 3 as part of V_{ERR}. The affects of these changes are discussed later.

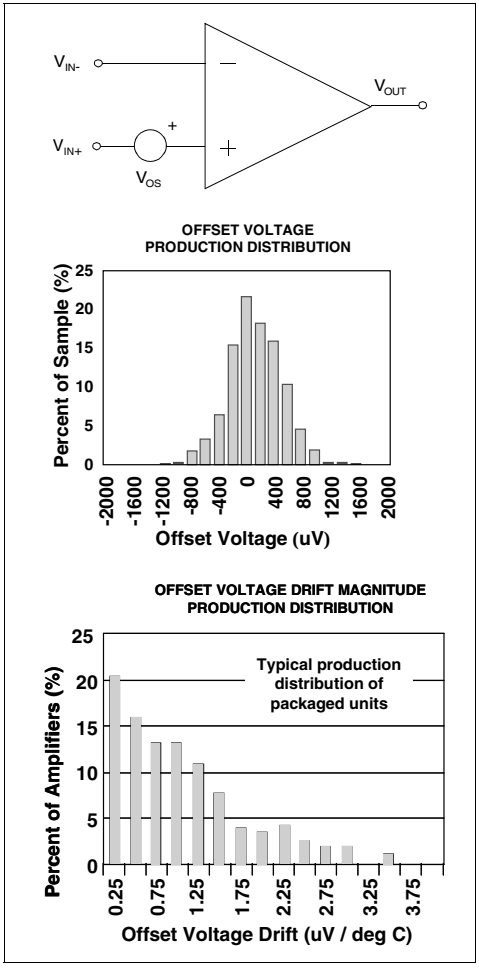


FIGURE 4: *The input offset voltage of an amplifier varies from part to part but always falls within the stated specification voltage range.*

Application Challenge - The offset voltage error of a particular amplifier may or may not be a problem, dependent on the application circuit. For instance, if a device is configured as a buffer (also known as a voltage follower), amplifiers with larger offset voltage errors, in the range of 2mV to 10mV, are usually not significantly different in performance than high precision amplifiers with extremely low offset voltage specifications, in the range of 100µV to 500µV. On the other hand, an amplifier with a high offset voltage that is in a high closed loop gain configuration can dramatically compromise the dynamic range of the circuit.

For example, the circuit in the Figure 5 is designed so that the analog input voltage (V_{IN}) is gained by:

$$V_{OUT} = (1 + R_F / R_{IN}) (V_{IN} + V_{OS})$$

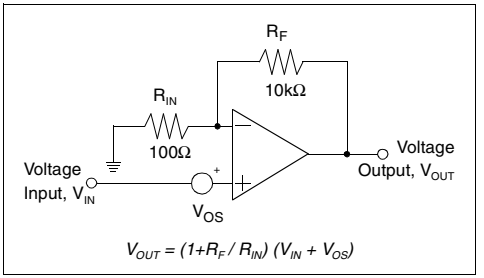


FIGURE 5: *An amplifier with a high input Offset Voltage can cause errors in the system, if the amplifier is configured in a high closed loop gain circuit.*

Unfortunately, the offset voltage of the amplifier is also multiplied by the same gain factor as the input signal. In this example, $(1 + R_F / R_{IN})$ is equal to 101V/V. An amplifier with an offset voltage of 1mV would produce a constant DC error at the output of 101mV. In a 5V system, 101mV lessens the dynamic range by approximately 2%.

Input Bias Current (I_B, I_{B+}, I_{B-}, and I_{OS})

Specification Discussion - All op amps have a leakage current that sources or sinks at both input terminals. Typically, this leakage current is called input bias current. The model for input bias current error is shown in Figure 3. The input offset current (I_{OS}) is equal to the difference between the input bias current at the non-inverting terminal (I_{B+}) minus the input bias current at the inverting (I_{B-}) terminal of the amplifier.

With CMOS and FET input amplifiers, the magnitude of the input bias current ranges from sub-pico amperes to several hundred pico amperes. The leakage at the input terminals of the CMOS amplifier usually does not come from the gate of the CMOS device but rather from the ESD cell. At room temperature, the input bias current of a CMOS amplifier can be less than a few tens of pico amperes. As the temperature increases, the ESD cells start to conduct current. This current appears at the input terminals of the amplifier.

In contrast, amplifiers with Bipolar inputs typically have input bias currents that range in the 10s of nano amps to several hundred nano amps. This current is the base current of the input Bipolar transistors. These amplifiers also have ESD cells, but the leakage from the base of the input transistor is much higher than the leakage from the ESD cells over temperature.

Application Challenge - Circuits that use high value resistors in the feedback loop or at the input of the amplifier are the most sensitive configurations for the op amp's input bias current error. For instance, if a high value resistor, such as 100kΩ is placed in series with the input of a Bipolar input amplifier that has an input bias current of 100nA, the resultant voltage is 100kΩ x 100nA or 10mV. This error at the input to the amplifier is added to any offset voltage error and then gained by the amplifier circuit.

In contrast, the input bias current of a CMOS amplifier could be 100pA. The voltage generated by the combination of this input bias current and a 100kΩ resistor is 10µV. In this scenario it is quite possible that the offset voltage error of the amplifier is greater than the error generated by the input bias current.

An example of a circuit that might use higher value resistors is a filter, such as the low pass filter shown in the Figure 6. In this circuit, the poles are established using the combination of resistors and capacitors. As the cut-off frequency of a low pass filter is decreased, the RC time constants that generate the poles increase. In the situation where a low frequency, low pass filter is required, it is easy enough to find higher value capacitors. However, if board real-estate is an issue, higher value resistors are a more economical alternative.

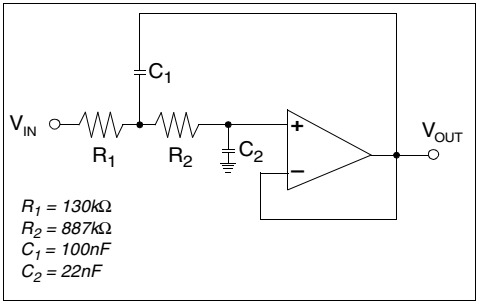


FIGURE 6: *This Sallen-Key, 2nd order, 10Hz, Butterworth, low pass filter circuit has two large resistors in series with the non-inverting input of the op amp. Input bias current errors from a Bipolar op amp will cause a considerable amount of error. In contrast, the input bias current from CMOS or BiFET amplifiers will be low enough not to cause appreciable errors.*

This RC relationship in combination with CMOS op amps can be used to an advantage with filters that have lower cut-off frequencies. Surface mount resistors can be found up to several mega ohms and surface mount film capacitors that are approximately the same size as the surface mount resistors can be found as high as several hundred nano farads. With this combination of passive devices, a compact, second order low pass filter can easily be designed down to 10Hz or lower.

In the example in Figure 6, a Bipolar amplifier with an input bias current of 100nA would generate a DC error through the resistor combination of R_1 and R_2 of 102.7mV. In contrast, a CMOS amplifier with an input bias current of 100pA would generate a DC error of 102.7µV.

Input Voltage Range (V_{IN} or V_{CM})

Specification Discussion - Each of the two input pins of the op amp has voltage swing restrictions. These restrictions are due to the input stage design. In the device product data sheet, the input voltage restrictions are clearly defined in one of two ways. Most commonly, the Input Voltage Range, V_{IN}, is specified as a separate line item in the specification table. This specification is also usually defined as a condition for the CMRR specification, input common-mode voltage range, V_{CM}. The more conservative specification of the two is where the input voltage range is called out as a CMRR test condition because the CMRR test validates the input voltage range with a second specification.

The input voltage range is more a function of the input circuit topology rather than the silicon process. Although the input devices of the amplifier can be CMOS, Bipolar or FET, there are three basic topologies that are used to design the input stage of single supply, voltage feedback amplifiers. These topologies are shown with a CMOS input stage in Figure 7. In Figure 7a, PMOS transistors (Q_1 and Q_2) are used for the first device at the input terminals. With this particular topology, the gate of both transistors can go 0.2 to 0.3V below the negative power supply voltage before these devices leave their active region. However, the input terminal can not go any higher than several hundred millivolts from the positive power supply voltage before the input devices are pulled out of their linear region. An amplifier designed with a PMOS input stage will typically have an input range of V_{SS} -0.2V to V_{DD} - 1.2V.

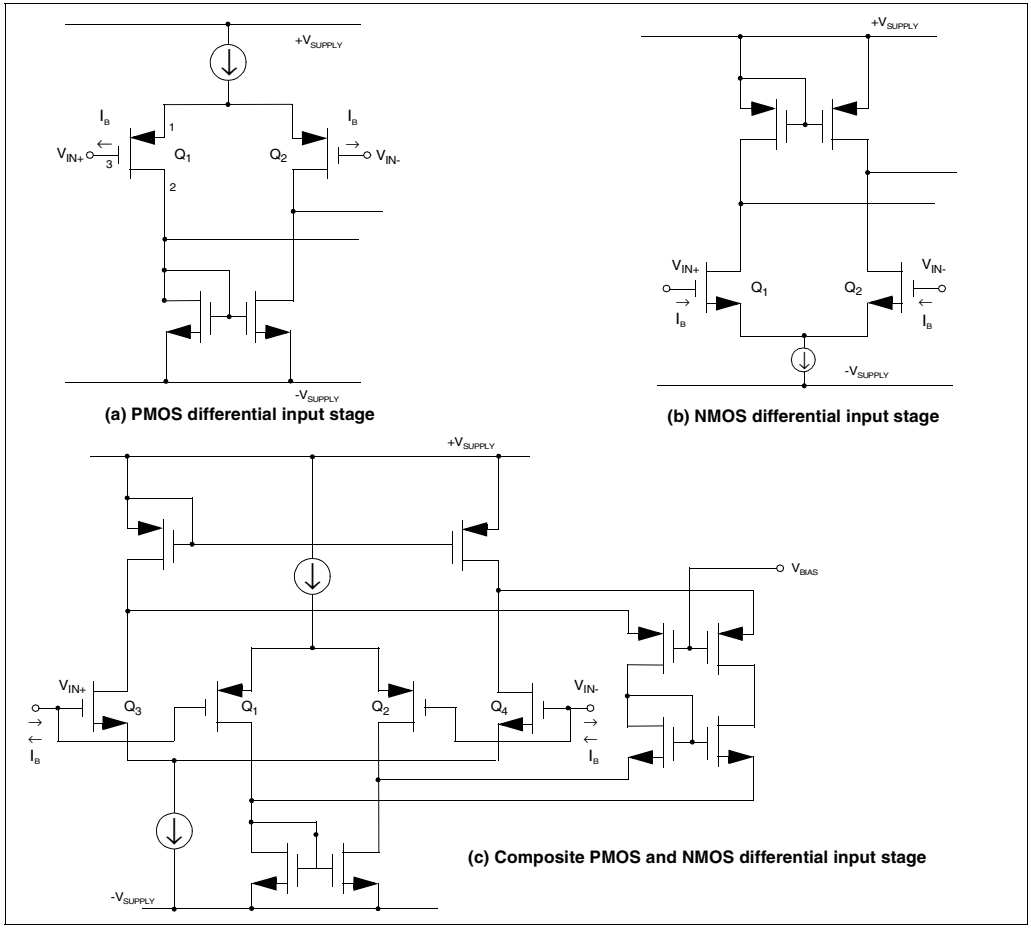


FIGURE 7: *The input voltage range of an op amp is dependant on the topology of the input stage of the amplifier. The input stage can be constructed of PMOS (a) devices allowing for the input to swing below the negative supply or a NMOS differential pair (b) where the inputs can swing above the positive supply. A composite input stage (c) uses PMOS and NMOS differential pairs so the input voltage range can extend from above the positive rail to below the negative rail.*

If the amplifier is designed with NMOS input transistors as shown in Figure 7b, the input range is restricted near the negative power supply voltage. In this case, the input terminals can be taken to a few tenths of a volt above the positive supply rail, but only to 1.2V above the negative supply rail.

If an amplifier input stage uses PMOS and NMOS transistors, it is configured as a composite stage, as shown in Figure 7c. With this topology, the amplifier effectively combines the advantages of the PMOS and NMOS transistors for true rail-to-rail input operation. When the input terminals of the amplifier are driven towards the negative rail, the PMOS transistors are turned completely on and the NMOS transistors are completely off. Conversely, when the input terminals are driven to the positive rail, the NMOS transistors are in use while the PMOS transistors are off.

Although, this style of input stage has rail-to-rail input operation there are trade-offs. This design topology will have wide variations in offset voltage. In the region near ground, the offset error of the PMOS portion of the input stage is dominant. In the region near the positive power supply, the input stage offset error is dominated by the NMOS transistor pair. With this topology, the offset voltage error can change dramatically in magnitude and sign as the common mode voltage of the amplifier inputs extend over their entire range.

The basic topologies shown in Figure 7 can be used with FET input or Bipolar input amplifiers. In the case of the FET input amplifier, the offset errors between the PFET and NFET are consistent with the CMOS errors with the circuit shown in Figure 7c. With Bipolar input stages, input offset voltage variations are still a problem, but input bias current is an additional error that is introduced. The nano ampere base current of an NPN transistor comes out of the device, while the nano ampere base current of a PNP transistor goes into the device.

Application Challenge - The input voltage range restrictions become critical in a subset of op amp circuit applications. For instance, if an op amp is configured as a voltage follower, it will most likely exhibit limitations in linearity due the input stage restrictions. This type of circuit is shown in Figure 8a, along with a current monitor circuit in Figure 8b.

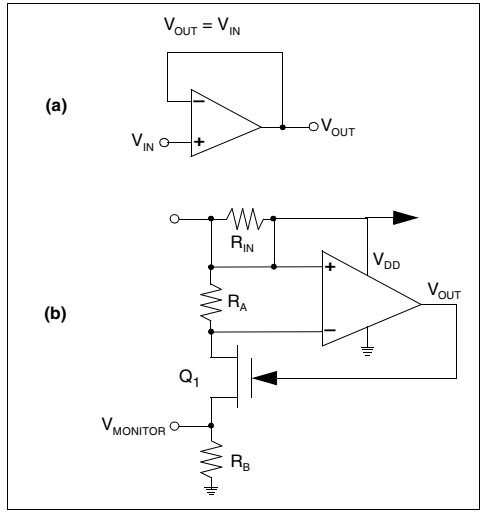


FIGURE 8: *If an amplifier is used as a buffer (a), the input devices of the operation amplifier may limit the input range of the buffer. If an amplifier is used in a high power supply sense circuit (b), the input stage must be able to extend to the positive rail.*

A buffer circuit configuration (Figure 8a.), requires rail-to-rail operation of the amplifier at its inputs as well as its outputs.

The high side current monitor circuit (Figure 8b), uses an op amp that must have an input voltage range up to the positive power supply rail. This circuit design senses the amount of current that is coming from the power supply. When the current exceeds 2A, the non-inverting input of the amplifier falls below the inverting input. As a result, that output goes low which turns off the JFET, Q_1, pulling the drain of the JFET low. This action brings the monitor output low.

These two applications present special requirements on the op amp. Most typically, an op amp is designed with a closed loop gain greater than one. In this instance, the output stage restrictions will limit the linear performance of the amplifier before the input stage will.

Open Loop Gain (A_{OL})

Specification Discussion - The Open Loop Gain of an op amp is the ratio of change in output voltage signal to the change in differential input voltage offset. This parameter is measured with or without a load. Ideally, the open loop gain of an amplifier should be infinite. In reality, the open loop gain, A_{OL}, is less than ideal at DC ranging from 95dB to 110dB. This can be translated into volts per volts with the formula:

$$A_{OL}\ (V/V) = 10^{(A_{OL}(dB)\ /\ 20)}$$

Using this formula, a 10µV differential input signal to an amplifier with an open loop gain of 100dB (10^5 V/V) in an open loop configuration would be gained to the output of the amplifier to 1V.

In production runs, the open loop gain can vary up to 30% from part to part, consequently, a closed loop system is a more desirable configuration when using an amplifier, unless the amplifier is used as a comparator. With a closed loop system, the gain is dependent on the accuracy of the resistors in the circuit.

In a closed loop system, the effects of the open loop gain error is easily determined with:

$$A_{OL}\ (dB) = 20\ log\ (\Delta V_{OUT} / \Delta V_{OS})$$

This formula states that a change in the output voltage of the closed loop system will generate a small change in offset voltage. The offset voltage error is then gained by the closed loop system, generating a gain error. (Refer to Figure 3, where ΔV_{OS} = open loop gain error.)

A load will degrade the open loop gain performance. Some manufacturers recognize this and specify more than one test condition.

Power Supply Rejection (PSRR)

Specification Discussion - The power supply rejection ratio specification quantifies the amplifier's sensitivity to power supply changes. Ideally, the power supply rejection ratio should be infinite. Typical specifications for a power supply rejection ratio of an amplifier range from 60dB to 100dB.

As is with the open loop gain (A_{OL}) characteristics of an amplifier, DC and lower frequency power supply noise is rejected more than at higher frequencies. In a closed loop system, a less than ideal power supply rejection capability of an amplifier manifests itself as an offset voltage error as shown in Figure 3 ($PSRR_{ERROR} = \Delta V_{OS}$). This error is best described with the formula:

$$PSRR(dB) = 20\ log\ (\Delta V_{SUPPLY} / \Delta V_{OS})$$

The formula that describes power supply rejection is:

$$PSR(V/V) = \Delta V_{OS} / \Delta V_{SUPPLY}$$

Where:

$$V_{SUPPLY} = V_{DD} - V_{SS}$$

Application Challenge - An application where power supply rejection is critical is shown in Figure 9. In this circuit, a battery is used to power an amplifier which is configured in a high, closed loop gain of 101V/V. During the life of the battery, the output voltage ranges from 5.75V down to 4.75V. If the power supply rejection of the amplifier is 500µV/V (or 66dB), the error at the output of the amplifier over time would be 50.5mV. In a 12-bit system with a full-scale range of 4.096V, this would equate to a 50.5 counts worth of offset change over the life of the battery.

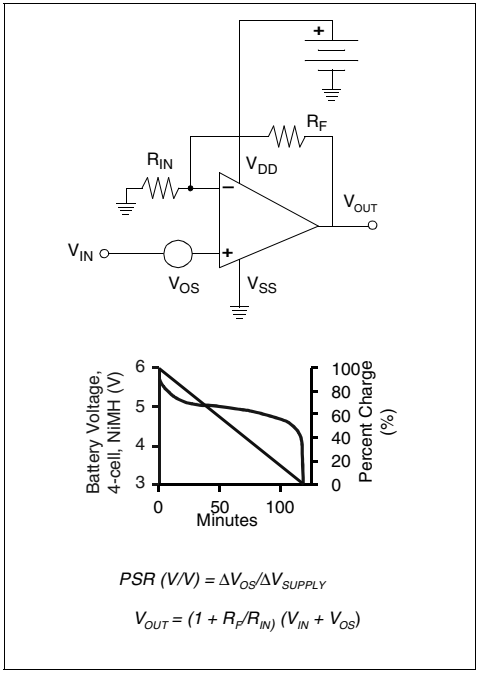


FIGURE 9: *A battery powered application can see a change in power supply voltage of several hundreds of millivolts over the life of the product. If an op amp is configured with a high closed loop gain in these types of applications, it must have good DC power supply rejection.*

Common Mode Rejection Ratio (CMRR)

Specification Discussion - The Common Mode Rejection Ratio of an amplifier describes the amplifier's input sensitivity to equivalent voltage changes of both inputs. This error manifests itself as an offset error ($CMRR_{ERROR}$), as shown in Figure 3.

$$CMRR(dB) = 20\ log\ (\Delta V_{CM} / \Delta V_{OS})$$

Where:

$$\Delta V_{OS} = CMRR_{ERROR}$$

Application Challenge - The specification range for CMRR in single supply amplifiers is from 45dB up to 90dB. Typically, this error becomes an issue when an amplifier is in a circuit where the input common mode voltage changes with input signal. A good example where this is the case, is when the amplifier is in a non-inverting configuration. A common circuit that has this configuration is shown in the Figure 10.

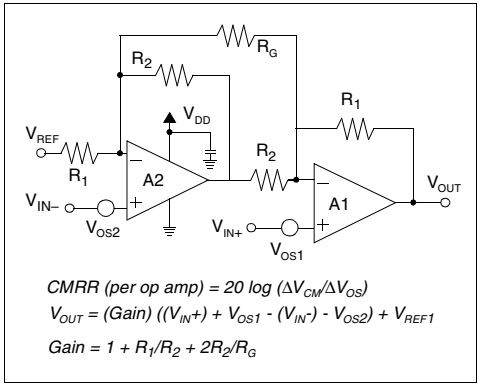


FIGURE 10: *A poor common mode rejection capability with either amplifier will cause an offset error that is gained to the output of the circuit.*

Voltage Output Swing (V_{OUT}, V_{OH}, or V_{OL})

Specification Discussion - The output swing specification of an op amp defines how close the output terminal of the amplifier can be driven to the negative or positive supply rail under defined operating and load conditions. Unlike the input voltage range (V_{IN}) specification, the voltage output swing of an amplifier is not as well defined from manufacturer to manufacturer. The output current as well as the amplifier's open loop gain (A_{OL}) are related to this specification. The output current is a test condition for the voltage output swing specification.

It is also a test condition for the open loop gain test, which validates the voltage output swing test with a second amplifier specification.

The output swing capability of the amplifier is dependent on the output stage design and the amount of current that the output stage is driving under test. With this portion of the specification, care should be taken when comparing amplifiers.

For instance, a single supply amplifier, MCP601, is used to generate the data in Table 1. It should be noted that the defined conditions of this specification have a significant influence on the amplifier's performance capability. All of these conditions, as well as others, can be found in op amp data sheets.

The key to comparing voltage output swing specifications, is to determine the amount of current that the amplifier is sinking or sourcing. The smaller the output current is, the closer the amplifier will swing to the rail.

If the load is specified as a current, this determination is easy. However, if the load is reference to $(V_{DD} - V_{SS})/2 + V_{SS}$, the output current is determined by dividing the voltage across the load resistor by the load resistor. It is useful to note that when the load is referenced to $(V_{DD} - V_{SS})/2 + V_{SS}$, the output of the amplifier will be sourcing or sinking half the current, as when the load is referenced to V_{DD} or V_{SS}.

The device in Table 1 was tested with the V_{DD} equal to 5V and V_{SS} equal to ground. Since this data was taken with one device, it does not necessarily represent the performance of all devices in the product family.

Output Voltage Swing	Test Conditions	Measured Output Swing from V_{SS} (mV)	Measured Output Swing from V_{DD} (mV)
High, to V_{DD}	w / 10kΩ load referenced to $(V_{DD} - V_{SS}) / 2 + V_{SS}$		11.2
High, to V_{DD}	w / 10kΩ load referenced to V_{SS}		20.4
High, to V_{DD}	w / 10kΩ load referenced to V_{DD}		1.95
High, to V_{DD}	w / amplifier source current equal to 100µA		3.8
Low, to V_{SS}	w / 10kΩ load referenced to $(V_{DD} - V_{SS}) / 2 + V_{SS}$	11.6	
Low, to V_{SS}	w / 10kΩ load referenced to V_{SS}	3.7	
Low, to V_{SS}	w / 10kΩ load referenced to V_{DD}	25.5	
Low, to V_{SS}	w / amplifier sink current equal to 100µA	8.1	

TABLE 1: *This data was taken with one sample of the MCP601 op amp and demonstrates the effects of the output conditions on the output swing performance of that amplifier. This data was taken with no regard to the open loop gain of the amplifier.*

The output voltage swing versus input offset voltage of this amplifier is shown in Figure 11. By using this plot, the open loop gain of the device can be calculated as the slope between two points. For example, the open loop gain of this amplifier using V_{OUT} = 1V to 4V, is 75dB.

With this plot, it is noticeable that the linearity of the amplifier starts to degrade long before the output swing maximums are reached. If the output of an amplifier is operated beyond the linear region of this curve, the input to output relationship of the signal will be non-linear.

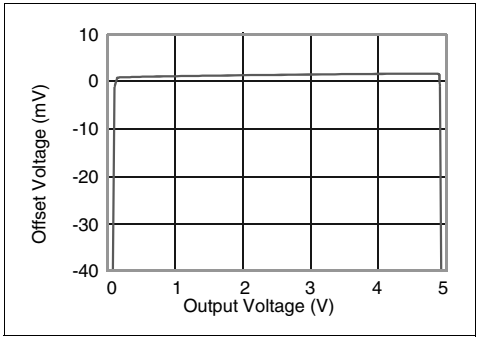


FIGURE 11: *This graph shows the relationship between the output swing of an amplifier and input offset voltage with a 25kΩ load and V_{DD} = 5V. The open loop gain of the amplifier can be calculated by selecting two points on the graph and calculating the slope. As the output swing of the amplifier goes towards the rail, the amplifier function eventually breaks down. This is first manifested with changes in input offset voltage.*

Output Impedance (R_{OUT}, R_{CL}, R_{OL}, Z_{CL}, Z_{OL})

Specification Discussion - The fact that the output impedance of an op amp is low, makes the device useful in terms of "isolating" the impedance of two portions of a circuit. For this reason, low output impedance of an op amp is an important characteristic, but the precise output impedance is usually is not specified.

When the output impedance is specified, it is given in terms of a resistance or impedance of a closed loop configuration (R_{CL} or Z_{CL}) or an open loop configuration (R_{OL} or Z_{OL}). Output impedance is most often specified as resistance.

Closed loop output resistance is the easiest to measure and is equal to:

$$R_{CL} = \Delta V_{OUT} / \Delta I_L$$

where

ΔV_{OUT} = the change in output voltage and

ΔI_L = the change in output current with a change in output voltage

The effective closed loop output impedance is less than the open loop output impedance by a factor equal to the reciprocal of the loop gain. The loop gain is equal to the open loop gain of the amplifier divided by the closed loop gain of the non-inverting circuit. For the circuit shown in Figure 12, the open loop resistance is equal to:

$$R_{CL} = R_{OL} / (A_{OL} / (1 + R_F / R_{IN}))$$

In this formula $(1 + R_F / R_{IN})$, is the non-inverting closed loop gain. This closed loop gain is also known as $1/\beta$.

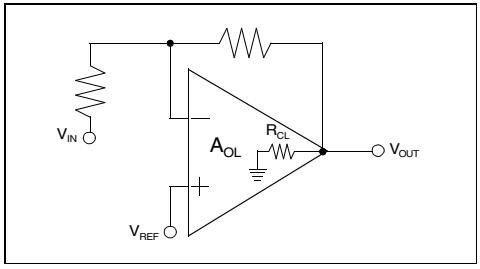


FIGURE 12: *The closed loop output resistance of an amplifier is lessened by the magnitude of the open loop gain of the amplifier.*

Power Supply Requirements (V_{SS}, V_{DD}, I_{DD}, I_Q)

Specification Discussion - Power supply voltage defines the acceptable difference between V_{DD} and V_{SS} which allows linear operation of the amplifier. If this voltage difference is less than specification, the amplifier may not operate reliably. If the power supply voltage is greater than specified, the amplifier most likely will operate as expected, but it is possible that damage may occur due to overvoltage stress on the internal transistors in the amplifier.

The power supply range is usually listed as a separate line item in the specification table in the product data sheet. Occasionally, the specification is called out as a condition under the PSRR specification.

Power supply current (I_{DD} or I_Q) is specified with no load. Typically, if a load is applied to the amplifier, a source current will primarily be pulled from V_{DD}, through the op amp output stage, and then through the load. A sink current will primarily result in an increase of V_{SS}.

Temperature Range

There are three types of temperature ranges that are specified with op amps.

- *Specified Temperature Range* - The range where the amplifier will meet specifications as called out in the specification table.
- *Operating Temperature Range* - The range where the amplifier will operate without damage but performance is not necessarily guaranteed.
- *Storage Temperature Range* - Defines the temperature maximums and minimums where the package will not sustain permanent damage. In this range the amplifier may not operate properly.

CONCLUSION

When searching for the right amplifier for an application, various performance specifications need to be taken into consideration. The first set of specifications to consider are the affects of the DC limitations of the amplifier. In single supply applications, amplifier errors such as input voltage swing, input offset voltage and input bias current could reduce the dynamic range of the amplifier. Conversely, in high gain circuits, the output voltage swing could cause signal clipping problems.

The second set of specifications to consider are the AC specifications. These issues are discussed in detail in the application note from Microchip entitled “Operational Amplifier AC Specifications and Applications”, AN723 (*available December, 1999.*)

REFERENCES

Wait, Huelsman, Korn, Introduction to Operational Amplifier Theory and Applications, *McGraw Hill, 1975*

“Operational Amplifier AC Specifications and Applications”, Baker, Bonnie, *Microchip Technology, Inc. AN723 (available December, 1999)*

SECTION 5 NON-VOLATILE MEMORY APPLICATION NOTES AND TECHNICAL BRIEFS

AN668

Interfacing a Microchip PIC16C92x to Microchip SPI™ Serial EEPROMs

Author: *Shannon Poulin*
Microchip Technology Inc.

INTRODUCTION

There are many different microcontrollers on the market today that are being used in embedded control applications. Many of these embedded control systems need non-volatile memory. Because of their small footprint, byte level flexibility, low I/O pin requirement, low power consumption, and low cost, serial EEPROMs are a popular choice for non-volatile storage.

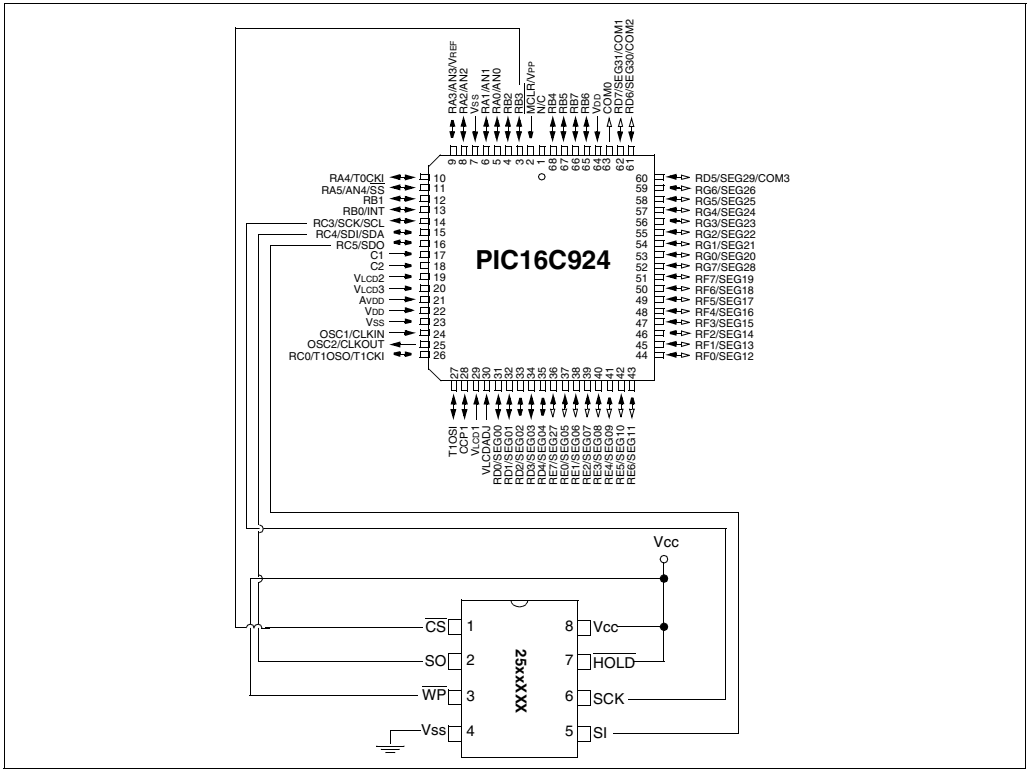
Microchip addresses these needs by offering a full line of serial EEPROMs covering industry standard serial communication protocols for two-wire, three-wire, and SPI communication. Serial EEPROM devices are available in a variety of densities, operational voltage ranges, and packaging options.

This application note provides assistance and source code to ease the design process of interfacing a Microchip PIC16C92x microcontroller and a Microchip SPI serial EEPROM. The hardware SPI port on the microcontroller provides a simple three-wire connection to the EEPROM and no external “glue” logic is required

Figure 1 describes the hardware schematic for the interface between Microchip’s SPI devices and the Microchip PIC16C92x Microcontroller. The schematic shows the connections necessary between the microcontroller and the serial EEPROM, and the software was written assuming these connections. Appendix A contains a listing of the SPI source code.

SPI is a trademark of Motorola, Inc.

FIGURE 1: CIRCUIT FOR PIC16C92X



Please check Microchip's Worldwide Website at www.microchip.com for the latest version of the source code.

APPENDIX A: SOURCE CODE

```
MPASM 01.40.01 Intermediate     ANXXX.ASM   4-7-1997  14:04:01         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST P=16C924
                     00002
                     00003 ; Port C pin descriptions
                     00004 ; SCK bit = 3
                     00005 ; SDI bit = 4
                     00006 ; SDO bit = 5
                     00007 ; CS bit  = 7
                     00008 ;
                     00009 ; 10MHz crystal is being used, thus each instruction cycle = 400nS
                     00010 ;
                     00011 ;******************RAM register definitions**********************
  00000020           00012 rxdata  equ     20h
  00000021           00013 txdata  equ     21h
  00000022           00014 addr    equ     22h
  00000023           00015 loops   equ     23h
  00000024           00016 outbyte equ     24h
  00000025           00017 temp1   equ     25h
  00000026           00018 temp2   equ     26h
                     00019 ;******************Bit definitions*******************************
                     00020
                     00021 #define   CS        3
                     00022 #define   SMP       7             ; SSPSTAT register bit definition
                     00023 #define   CKE       6             ; SSPSTAT register bit definition
                     00024 #define   SPI_HOLD  PORTB,4       ; SPI Hold pin definition
                     00025 #define   SPI_WP    PORTB,1       ; SPI Write Protect pin definition
                     00026 #define   SPI_CS    PORTB,3       ; SPI Chip Select bit definition
                     00027
                     00028 ;******************Include file*********************************
                     00029         include "p16c924.inc"
                     00001         LIST
                     00002 ; P16C924.INC Standard Header File, Version 1.00 Microchip Technology, Inc.
                     00288         LIST
                     00030 ;*****************************************************************
0000                 00031         org     0x000           ; set the reset vector
0000 2801            00032         goto    start           ; go to the beginning of main
                     00033
                     00034 ;!!!!!!!!!!!!!!!!!!!!!!Begin Main Program!!!!!!!!!!!!!!!!!!!!!!!!
0001 1283            00035 start   bcf     STATUS,RP0      ; set to bank 0
0002 0187            00036         clrf    PORTC           ; initialize portc to 0
0003 1683            00037         bsf     STATUS,RP0      ; set to bank 1
0004 3010            00038         movlw   0x10            ; all bits are outputs except SDI
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0005 0087            00039         movwf   TRISC           ; move the value to TRIS portc
                     00040 ;***
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0006 0186            00041         clrf   TRISB
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0007 1394            00042         bcf   SSPSTAT,SMP
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0008 1314            00043         bcf   SSPSTAT,CKE
0009 1283            00044         bcf     STATUS,RP0      ; set to bank0
                     00045 ;***
000A 1606            00046         bsf  SPI_HOLD
000B 1486            00047         bsf  SPI_WP
000C 1586            00048         bsf  SPI_CS
                     00049
```

```
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
000D 018C     00050         clrf    PIE1            ; disable peripheral interrupts
000E 018B     00051         clrf    INTCON          ; disables all interrupts
000F 1283     00052         bcf     STATUS,RP0      ; set to bank 0
0010 0194     00053         clrf    SSPCON          ; clear SSP control register
0011 3031     00054         movlw   0x31            ; set up spi port, SPI master,
0012 0094     00055         movwf   SSPCON          ; clk/16, ckp=1 (mode 1,1)
0013 1587     00056         bsf     PORTC,3         ;bug workaround
0014 3055     00057         movlw   0x55            ; put starting address 55 in
0015 00A2     00058         movwf   addr            ; addr for later use
              00059 ;Send the write enable sequence (WREN)
0016 1186     00060         bcf     SPI_CS          ; clear the chip select line
0017 3006     00061         movlw   0x06            ; load WREN sequence
0018 00A4     00062         movwf   outbyte         ; store in RAM location outbyte
0019 2065     00063         call    output          ; call the SPI output routine
001A 1586     00064         bsf     SPI_CS          ; set the chip select line
              00065 ;Send the write status register sequence (WRSR)
001B 1186     00066         bcf     SPI_CS          ; clear the chip select line
001C 3001     00067         movlw   0x01            ; clear all status register
001D 00A4     00068         movwf   outbyte         ; store in RAM location outbyte
001E 2065     00069         call    output          ; call the SPI output routine
001F 3000     00070         movlw   0x00            ; load up zero to send
0020 00A4     00071         movwf   outbyte         ; store in RAM location outbyte
0021 2065     00072         call    output          ; call the SPI output routine
0022 1586     00073         bsf     SPI_CS          ; set the chip select line
              00074 ;Wait the required 5mS for the write cycle timer Twc
0023 2070     00075         call    delay           ; call the delay subroutine
              00076 ;Send the write enable sequence (WREN)
0024 1186     00077         bcf     SPI_CS          ; clear the chip select line
0025 3006     00078         movlw   0x06            ; load WREN sequence
0026 00A4     00079         movwf   outbyte         ; store in RAM location outbyte
0027 2065     00080         call    output          ; call the SPI output routine
0028 1586     00081         bsf     SPI_CS          ; set the chip select line
              00082 ;Send the read status register sequence (RDSR)
0029 1186     00083         bcf     SPI_CS          ; clear the chip select line
002A 3005     00084         movlw   0x05            ; load RDSR sequence
002B 00A4     00085         movwf   outbyte         ; store in RAM location outbyte
002C 2065     00086         call    output          ; call the SPI output routine
002D 2065     00087         call    output          ; read the data in status reg.
002E 1586     00088         bsf     SPI_CS          ; set the chip select line
              00089 ;Send the write sequence (WRITE)
002F 1186     00090         bcf     SPI_CS          ; clear the chip select line
0030 3002     00091         movlw   0x02            ; load WRITE sequence
0031 00A4     00092         movwf   outbyte         ; store in RAM location outbyte
0032 2065     00093         call    output          ; call the SPI output routine
              00094 ;****Comment out for use with 25xx010, 25xx020 or 25xx040********
0033 3000     00095         movlw   0x00            ; load high address byte
0034 00A4     00096         movwf   outbyte         ; store in RAM location outbyte
0035 2065     00097         call    output          ; call the SPI output routine
              00098 ;*************************************************************
0036 0822     00099         movf    addr,W          ; move the address into w
0037 00A4     00100         movwf   outbyte         ; store in RAM location outbyte
0038 2065     00101         call    output          ; call the SPI output routine
0039 30AA     00102         movlw   0xAA            ; load data AA into w
003A 00A4     00103         movwf   outbyte         ; store in RAM location outbyte
003B 2065     00104         call    output          ; call the SPI output routine
003C 1586     00105         bsf     SPI_CS          ; set the chip select line
              00106 ;Perform data polling (RDSR bit 1)
003D 1186     00107         bcf     SPI_CS          ; clear the chip select line
003E 3005     00108         movlw   0x05            ; load RDSR sequence
003F 00A4     00109         movwf   outbyte         ; store in RAM location outbyte
0040 2065     00110         call    output          ; call the SPI output routine
              00111 ;
0041 3030     00112         movlw   0x30            ; give the spi device time to
0042 00A3     00113         movwf   loops           ; set the WIP bit in the
Message[305]: Using default destination of 1 (file).
```

```
0043 0BA3     00114 wait    decfsz  loops           ; status register before coming
0044 2843     00115         goto    wait            ; back and doing data polling
              00116 ;
0045 2065     00117 polling call    output          ; read the data in status reg.
0046 1820     00118         btfsc   rxdata,0        ; test the WIP bit in status reg.
0047 2845     00119         goto    polling         ; WIP clear, loop until WIP set
0048 1586     00120         bsf     SPI_CS          ; set the chip select line
              00121 ;Send read sequence (READ), read address 0x55
0049 1186     00122         bcf     SPI_CS          ; clear the chip select line
004A 3003     00123         movlw   0x03            ; load READ sequence
004B 00A4     00124         movwf   outbyte         ; store in RAM location outbyte
004C 2065     00125         call    output          ; call the SPI output routine
              00126 ;****Comment out for use with 25xx010, 25xx020 or 25xx040********
004D 3000     00127         movlw   0x00            ; load high address byte
004E 00A4     00128         movwf   outbyte         ; store in RAM location outbyte
004F 2065     00129         call    output          ; call the SPI output routine
              00130 ;*************************************************************
0050 0822     00131         movf    addr,W          ; move the address into w
0051 00A4     00132         movwf   outbyte         ; store in RAM location outbyte
0052 2065     00133         call    output          ; call the SPI output routine
0053 2065     00134         call    output          ; call output to read 1 byte
0054 1586     00135         bsf     SPI_CS          ; set the chip select line
              00136 ;Send the write enable sequence (WREN)
0055 1186     00137         bcf     SPI_CS          ; clear the chip select line
0056 3006     00138         movlw   0x06            ; load WREN sequence
0057 00A4     00139         movwf   outbyte         ; store in RAM location outbyte
0058 2065     00140         call    output          ; call the SPI output routine
0059 1586     00141         bsf     SPI_CS          ; set the chip select line
              00142 ;Send the write disable sequence (WRDI)
005A 1186     00143         bcf     SPI_CS          ; clear the chip select line
005B 3004     00144         movlw   0x04            ; load WRDI sequence
005C 00A4     00145         movwf   outbyte         ; store in RAM location outbyte
005D 2065     00146         call    output          ; call the SPI output routine
005E 1586     00147         bsf     SPI_CS          ; set the chip select line
              00148 ;Send the read status register sequence (RDSR)
005F 1186     00149         bcf     SPI_CS          ; clear the chip select line
0060 3005     00150         movlw   0x05            ; load RDSR sequence
0061 00A4     00151         movwf   outbyte         ; store in RAM location outbyte
0062 2065     00152         call    output          ; call the SPI output routine
0063 2065     00153         call    output          ; read the data in status reg.
              00154 ;*******Go back to main routine*********************************
0064 2801     00155         goto    start
              00156
              00157 ;******************SPI output subroutine***********************
0065 0824     00158 output  movf    outbyte,W       ; move outbyte into w
0066 0093     00159         movwf   SSPBUF          ; place data in send buffer
0067 1683     00160 loop1   bsf     STATUS,RP0      ; set to bank 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0068 1C14     00161         btfss   SSPSTAT,BF      ; has data been received?
0069 2867     00162         goto    loop1           ; loop if not received yet
006A 1283     00163         bcf     STATUS,RP0      ; set to bank 0
006B 0813     00164         movf    SSPBUF,W        ; empty the receive buffer
              00165
006C 1294     00166         bcf     SSPCON,SSPEN    ; disable SPI peripheral
006D 1694     00167         bsf     SSPCON,SSPEN    ; enable  SPI peripheral
              00168                                 ; the previous 2 bcf and bsf instructions are
              00169                                 ; are required per device errata.
006E 00A0     00170         movwf   rxdata          ; put received byte into rxdata
006F 3400     00171         retlw    0              ; return from subroutine
              00172
              00173 ; 250 x 400nS x 50 = 5mS (plus overhead)
0070 3032     00174 delay   movlw   0x32            ; move 50 decimal into w
0071 00A5     00175         movwf   temp1           ; move 50 decimal into temp1
0072 30FA     00176 dec1    movlw   0xFA            ; move 250 decimal into w
0073 00A6     00177         movwf   temp2           ; move 250 decimal into temp2
0074 0BA6     00178 dec2    decfsz  temp2,1         ; decrement temp2, skip if zero
```

```
0075 2874     00179         goto    dec2            ; goto decrement 2 if not zero
0076 0BA5     00180         decfsz  temp1,1         ; decrement temp1, skip if zero
0077 2872     00181         goto    dec1            ; goto decrement 1 if not zero
0078 3400     00182         retlw   0               ; return both locations = 0
              00183
              00184         END


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXX-------

All other memory blocks unused.

Program Memory Words Used:   121
Program Memory Words Free:  3975


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     7 reported,     0 suppressed
```

MICROCHIP

AN671

Converting from 93LC56/56B/66/66B Devices to 93LC56A/56B/66A/66B Devices

Author: *Shannon Poulin*
Microchip Technology Inc.

DESCRIPTION

This application note details the process of converting from 93LC56, 93LC56B, 93LC66, and 93LC66B type devices to Microchip's new 93LC56A, 93LC56B, 93LC66A, and 93LC66B devices. The new devices offer improved data polling, lower standby current, lower operating current and are available in a small 8-pin TSSOP package. The new devices also offer fixed device organization. For example, the "A" suffix on the 93LC56A indicates that the device is organized as x8 only. The "B" suffix indicates that the 93LC56B is organized as x16 only.

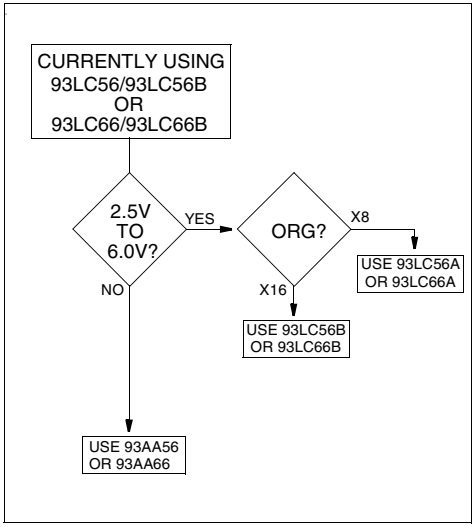
The new devices also incorporate power protection circuitry that adds extra data protection when powering the device up and down. The 93LCxxA/B devices are designed to function as 2.5V-6.0V parts. The internal voltage-detect circuit inhibits writes at < 2.2V nominally.

The operational parameters of the old and new devices are outlined below.

Part Number	Organization	Operating Range
93LC56	x8 or x16	2.0V-6.0V
93LC56B	x16 only	2.0V-6.0V
93LC66	x8 or x16	2.0V-6.0V
93LC66B	x16 only	2.0V-6.0V
93AA56	x8 or x16	1.8V-6.0V
93AA66	x8 or x16	1.8V-6.0V
93LC56A **NEW**	x8 only	2.5V-6.0V
93LC56B **NEW**	x16 only	2.5V-6.0V
93LC66A **NEW**	x8 only	2.5V-6.0V
93LC66B **NEW**	x16 only	2.5V-6.0V

The following product conversion decision tree will assist in converting from the old to the new devices. In order to properly convert to new products two items need to be known about the application; the operating voltage of the system and whether the memory is organized as x8 or x16.

PRODUCT CONVERSION DECISION TREE



NOTES:

MICROCHIP

AN674

Solving Second Sourcing Issues with the 24LC00 Device in a SOT-23 Package

Author: *Keith Pazul*
Microchip Technology Inc.

INTRODUCTION

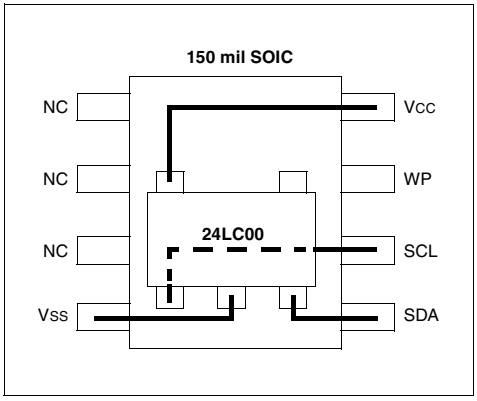
Many potential customers of the 24LC00 have expressed interest in the device, especially with it available in the ultra-small SOT-23 package. The one thing that has kept customers from designing in or qualifying the 24LC00 is the fact that it is a proprietary device without a second source. Because of this issue, Microchip Technology Inc. has put together this application note to assist customers in designing their applications to except a 24LC00 in a SOT-23 or a standard 1K-bit I^2C device in an 8-lead, 150 mil SOIC package.

HARDWARE/LAYOUT CONSIDERATIONS

Figure 1 below shows a board layout having both packages to scale with respect to each other. One can see from the figure that if the 24LC00 is placed as shown, a board can be easily designed to accept both devices. The connection from SCL on the 24LC00 to the SCL pad location on a standard I^2C device is shown as a dashed line as it runs underneath the 24LC00.

Since the 24LC00 does not have a WP pin, no connection was made from the 24LC00 to the WP pin of a standard I^2C device. Most devices that contain WP pins require it to be tied to either V_{CC} or V_{SS}, so a trace running to the WP pad will required for most applications.

FIGURE 1: BOARD LAYOUT ALLOWING SECOND SOURCING OF MICROCHIP'S 24LC00



SOFTWARE CONSIDERATIONS

The Microchip 24LC00 is the only 16 byte device on the market that has a full function I^2C interface. That means standard software routines that have been written for I^2C communication will work with the 24LC00 device. In addition, Microchip offers I^2C communication application notes and source code available for download on our web site (www.microchip.com).

The 24LC00 does not include page-write capability found on most of the industry's 1K-bit and higher I^2C Serial EEPROMs. Software written for the 24LC00 will be able to be utilized without modification in systems where an industry standard 1K I^2C Serial EEPROM is used. The one caveat is that the 24LC00 will answer to any I^2C memory address from 000 to 111. Some 1K I^2C Serial EEPROMs in the market have active address inputs, and therefore answer only to the address that is specified by the state of the three address inputs. Care must be taken to make sure address inputs match the application software if a second source device with active address inputs is chosen.

NOTES:

MICROCHIP

AN676

Physical Slot Identification Techniques for the 24LCS61/62

Author: *Rick Stoneking*
Microchip Technology Inc.

INTRODUCTION

With the complexity of systems ever increasing, system designers are faced with new challenges in identifying and assigning system resources on-the-fly. Often a system will have multiple peripheral cards that must be added on demand to fill a processing requirement or configuration. These cards will need to be dynamically identified and added to the system's resource management database.

Various Plug and Play (PnP) solutions have been developed to meet these requirements. A common thread among PnP schemes is to have a lower bandwidth channel, such as the I^2C bus, allocated as a means of interrogating newly added system cards. Through this data link, the system master can get or assign crucial information and calibration data from a specific card, without impacting the system's higher bandwidth bus.

SOFTWARE ADDRESSABLE SOLUTIONS

The Microchip Technology Inc. 24LCS61/62 is a 1K/2K bit Serial EEPROM developed for applications that require non-volatile storage of data and to have many devices on the same bus but do not have the spare I/O pins required to address each device individually. These devices contain an 8-bit address register that is set upon power-up and allows the connection of up to 255 devices on the same bus. When the process of assigning ID values to each device is in progress, the device will automatically handle bus arbitration if more than one device is operating on the bus. In addition, an external open drain output pin ($\overline{EDS}$) is available that can be used to enable other circuitry associated with each individual system. See the Microchip 24LCS61/62 (DS21226) data sheet for further information on these devices.

A PNP SCENARIO APPLICATION

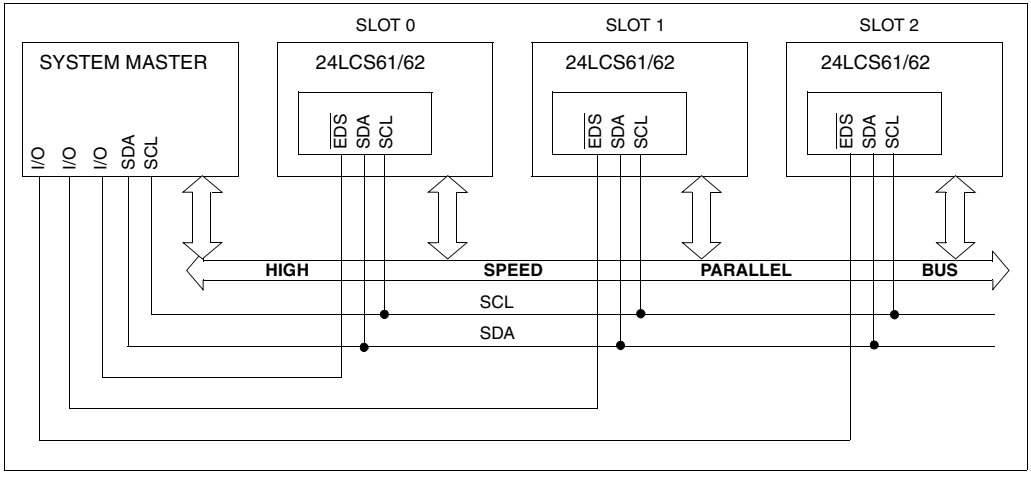
This application will utilize a system made of multiple circuit cards plugged into a backplane. A common high bandwidth bus and an I^2C™ bus will interconnect the slots of the backplane. The system master must create a table that correlates the card type inserted into each slot. To do this, the system master needs to read the information about each board in the system. This information is stored in the 24LCS61/62 on that board. The system master also needs to determine the physical slot number that the board is plugged into. Listed below are five methods for determining physical slot locations.

Method 1: Direct Connection of the $\overline{EDS}$ Pins to the System Master

The simplest method to determine the physical slot location is to route the $\overline{EDS}$ output pin from each card back to the system master. This physical connection, or return line, serves as a fixed slot identifier. Each device on the bus would then be identified by connecting this line to a unique input pin on the system master. After performing the arbitration process, the system master can toggle the output of the $\overline{EDS}$ pin on a particular device, and sense the resulting physical slot location indicated by poling the I/O pins.

This method requires equal number of return lines as there are devices on the bus.

FIGURE 1: METHOD 1



I²C is a trademark of Philips Corporation.

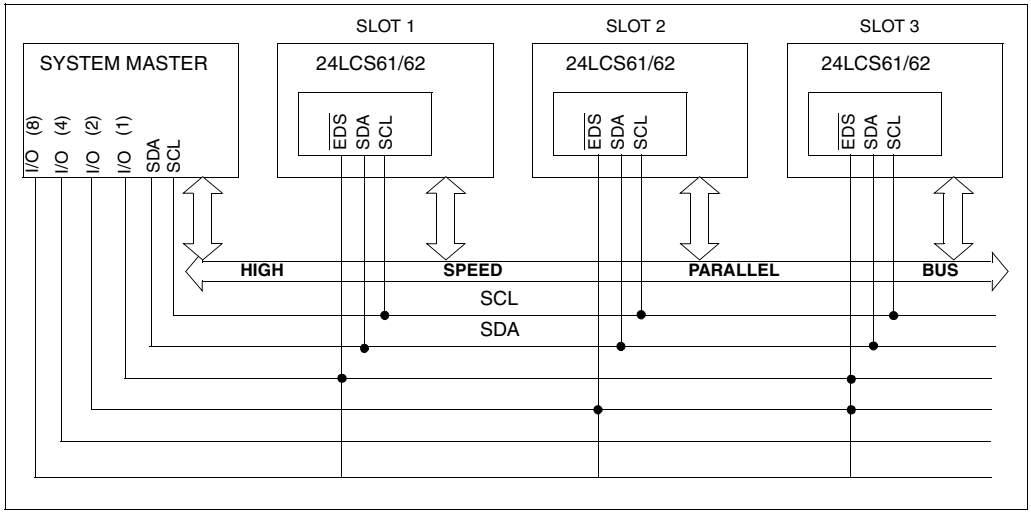
Method 2: Encoded Return Lines to System Master

Dedicated return lines can be encoded to identify physical slots. The encoding is made by the connections, or lack of connections, between the $\overline{EDS}$ pin on a particular device, and the return lines going back to the system master. These connections take place on the main bus backplane. After arbitration, the system master would activate the $\overline{EDS}$ pin on a single device in sequence, and read the encoded value on the return lines connected to the I/O pins. The physical slot identifier will be returned in binary coded form. In the example shown below, asserting the $\overline{EDS}$ pin on the device in slot 1 will yield the binary "0001", indicating to the master that this device is in slot 1. In this method, N return lines can encode a maximum of 2^N devices on the bus.

This method has the advantage that it only consumes one pin on the connector between the board and the backplane. Therefore, the entire bus to identify and interrogate boards in the system will only require three pins (SCL, SDA and $\overline{EDS}$) on the backplane connector.

For further reliability assurance, the designer can drop the code which has no physical connections to any return lines (i.e., "0000"). This would make every valid slot identifier assert at least one return line. This eliminates the possibility of a faulty device being erroneously assigned the '0' slot identifier. This lowers the total allowable number of slots given N return lines to $2^N - 1$.

FIGURE 2: METHOD 2



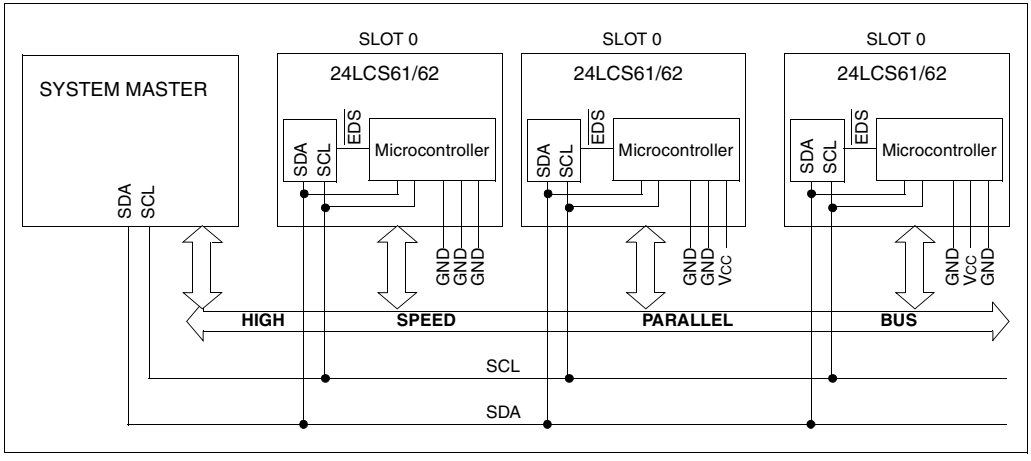
Method 3: Microcontroller Slot Address Sensing, and Serial Communication

Each card contains a 24LCS61/62 nonvolatile memory device, and a dedicated microcontroller (this example uses the Microchip PIC16C57) for transmitting slot information serially over the I^2C bus. Both are connected to the backplane's I^2C bus. The microcontroller on the card uses two dedicated I/O lines for SDA and SCL connections. This provides a feedback path for determining the physical slot in which a card resides. Other dedicated I/O lines by the microcontroller to sense a slot number from the dedicated connector pins. These pins are encoded with the slot identifier on the backplane.

First, the system master processor will initiate the 24LCS61/62 arbitration process until all memory devices are assigned unique ID. Next, the system master will sequentially address each 24LCS61/62 and assert the $\overline{EDS}$ pin. The $\overline{EDS}$ line serves as a select line for that particular microcontroller, and puts that microcontroller into a state where it will respond to commands from the system master. The microcontroller can wake up to a normally formatted I^2C command and respond with the physical slot identifier. Because the memory devices on the bus wake up to 06XH, and the microcontroller wakes up to 0AXH, there will never be a bus contention issue between them.

The number of I/O lines dedicated to sensing physical slot information will determine the number of devices addressable on the bus. If the microcontroller has eight I/O lines then the full number (255) of 24LCS61/62's can be addressed.

FIGURE 3: METHOD 3

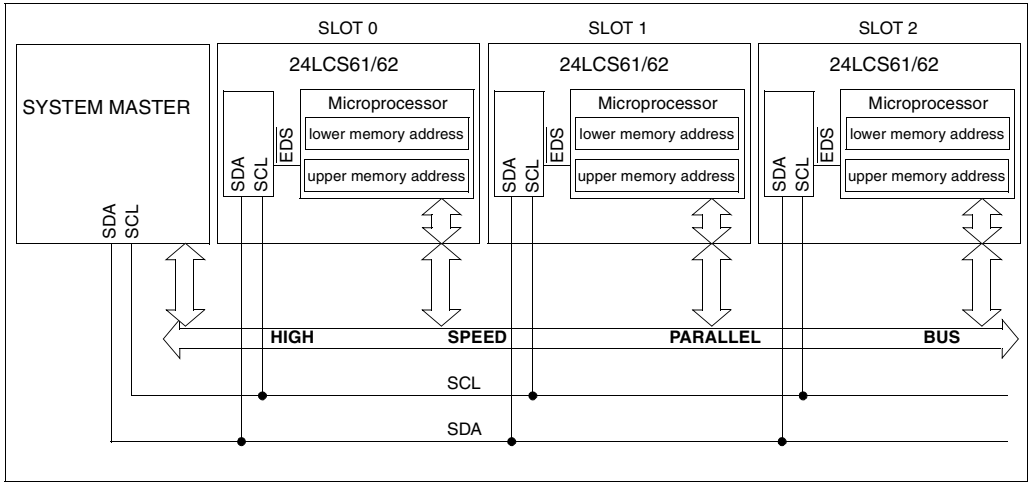


Method 4: Virtual Memory Addressing on Cards

This method is applicable when all communication between boards is accomplished by transfers between memory spaces on each board. In this scenario, the system master starts by querying the I^2C bus looking for unassigned 24LCS61/62 devices. This is done by repeatedly sending the assign address command and checking for acknowledging devices. If a device acknowledges, then a software address is assigned to that device after the arbitration cycle has been completed. The system master can then either continue assigning software addresses to any remaining 24LCS61/62 devices on the bus, or it may immediately assign a "virtual memory address" to the board associated with the just assigned 24LCS61/62 device (hereafter referred to as the "slave board"). The first step of assigning the virtual address is to read the configuration data from the 24LCS61/62 device. This tells the system master the amount of virtual memory space to reserve for the slave board. The system master would then issue a command to the 24LCS61/62 device with the OE bit set to '1', which enables the $\overline{EDS}$ pin on the 24LCS61/62 device. The $\overline{EDS}$ pin is tied to an input on the slave board microcontroller which signals it to enter a "receive virtual address" mode. Once the slave board is ready to receive the virtual address the system master sends the necessary information out over the main system bus to be received by the current slave board. Any necessary communication between the system master and the slave board is completed and the system master then issues a command to the 24LCS61/62 device with the OE bit set to a '0', which signals the slave board microcontroller to terminate the assign virtual memory address mode. This sequence is then repeated until no unassigned 24LCS61/62 devices remain on the I^2C bus. Once the process is complete the slave board can be addressed by its virtual address.

This method has the advantage of requiring only one I/O pin on the slave board microcontroller, and may be used when it is not necessary to know which physical slot a slave board is in.

FIGURE 4: METHOD 4



Method 5: Microcontroller Slot Address Sensing and Communication over System Bus

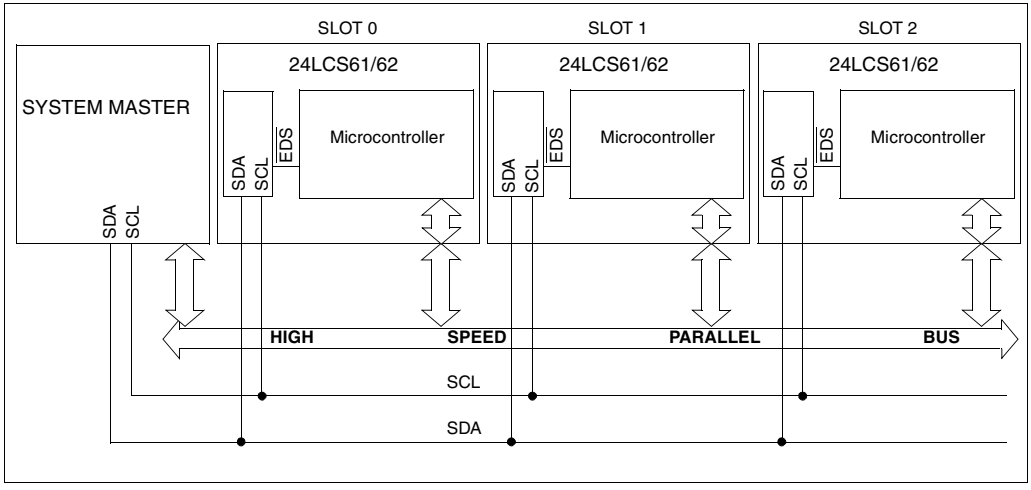
This method may be used if the physical slot must be identified and thus the slot ID number is hard-coded into the slave board connector. In this scenario the $\overline{EDS}$ pin of the 24LCS61/62 device is used to enable the slave board microcontroller in the same way as in the method above, however, once enabled, the slave board microcontroller reads the hard coded physical slot number and then sends this information to the system master via the system bus. From that point on the slave board can be addressed by using its slot ID number.

This method again requires only a single I/O on the slave board microcontroller, and is effective when the physical location of a given slave board must be known.

CONCLUSION

The Microchip 24LCS61/62 is a powerful building block in creating a board identification solution for multiboard systems. The methods outlined above are generic in nature, and meant to serve as a starting point for development of individual solutions for physical slot identification.

FIGURE 5: METHOD 5



MICROCHIP

AN683

How to Use the 24LCS61/62 Software Addressable Serial EEPROM

Author: *Rick Stoneking*
Microchip Technology Inc.

INTRODUCTION

The purpose of this application note is to provide an example of how to use the Microchip Technology 24LCS61/62 in a multi board type of system. A hypothetical system is presented and described, along with the source code to implement all of the features of the these devices.

SYSTEM DESCRIPTION

The hypothetical system described in this application note is an industrial controller that can be configured by adding/removing/changing individual circuit cards. This system utilizes a high speed parallel bus for normal communication between the individual cards, and an I^2C bus is used for detecting the installed system cards, determining their configuration, and initializing the system accordingly. After the initial power up sequence determines the 'boot configuration', this secondary bus is also periodically polled looking for new devices/cards.

HARDWARE DESCRIPTION

Figure 1 shows a block diagram of the system. This application note does not address the entire system, but only those portions necessary to demonstrate the code required for a typical implementation using the 24LCS61/62 devices. The system consists of a system master which controls the I^2C bus, and any number of additional circuit cards.

Each of these circuit cards has either a 24LCS61 or 24LCS62 on it, which allows the system master to identify the presence of each card, determine the type of card (e.g. Memory, I/O, Display), and read/write any necessary configuration data. Each circuit card may or may not have a microcontroller of it's own. For those that do the $\overline{EDS}$ pin of the 24LCS61/62 device is used as an input to the microcontroller to tell it that it has been detected by the system master.

SOFTWARE DESCRIPTION

Figure 2 shows the flow chart for the system master node initialization. At the start of the initialization the system master enters a software loop which sends the Assign ID command over the I^2C bus and looks for an acknowledge (ACK) signal, indicating that there is an unassigned device. If the ACK is detected the system master then reads the six byte device serial number which latches the assigned ID into the addressed 24LCS61/62 device. The serial number serves as an arbitration mechanism only in this system, nothing further is done with it after it is read. The system master then reads the first byte of the 24LCS61/62 EEPROM memory, which is defined to contain the size in bytes of the EEPROM array. This is so that the system master can write a time stamp to the last three bytes of the EEPROM after configuration has been completed. During this read the system master sets the $\overline{EDS}$ pin of the 24LCS61/62 low to signal the microcontroller (if any) on the circuit card that it has been detected and assigned.

FIGURE 1: SYSTEM BLOCK DIAGRAM

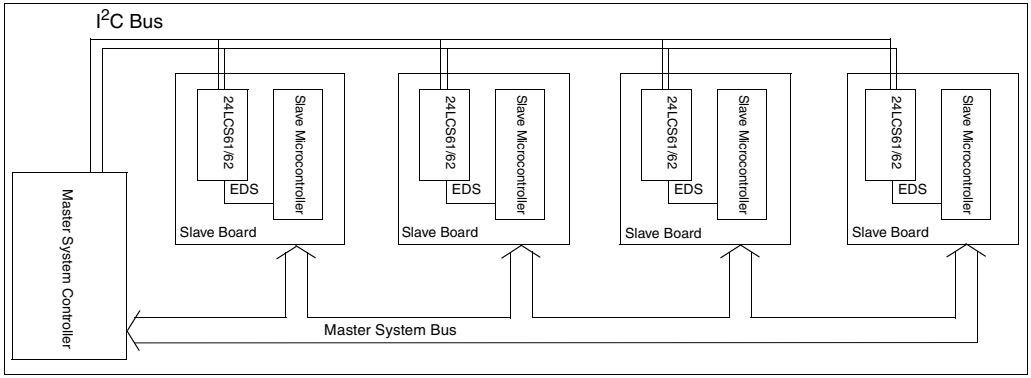
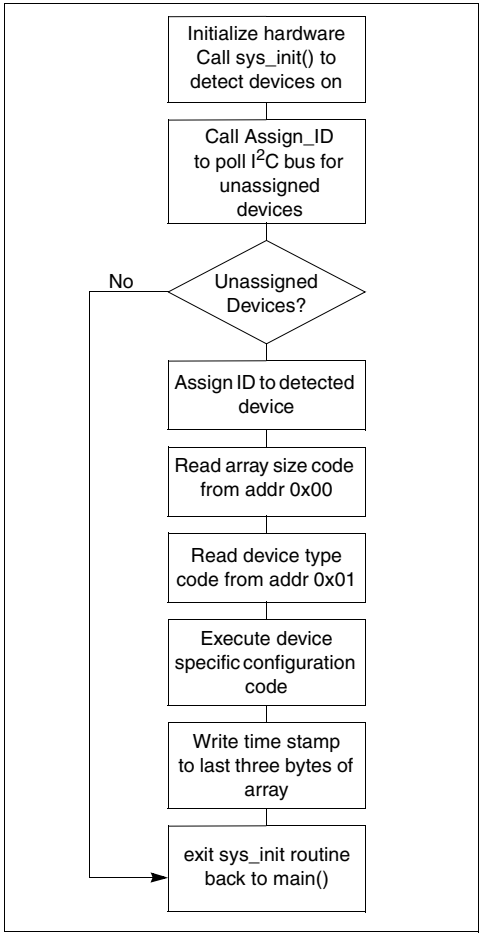


FIGURE 2: SYSTEM FLOW FOR MASTER NODE INITIALIZATION



SOURCE CODE DISCUSSION

Appendix A contains the 'C' source code for the functions necessary to perform the operations described in the Software Description section above. These functions rely on additional I^2C functions that are defined in the sw12c16.lib library created by, and available from, Microchip Technology. A listing for this library has not been included in this application note.

The following files are required in order to complete the source code in Appendix A:

File Name	Description
24lcs6x.c	Main file (listing 1)
swi2c16.h	MPLAB-C header file
swi2c16.lib	MPLAB-C library file
delays.h	MPLAB-C header file
delays.lib	MPLAB-C library file
math.h	MPLAB-C header file
17c42a.h	MPLAB-C header file

The source code was compiled using MPLAB-C v1.21 and has been verified to work. The source code, including the required libraries and header files, is available for download from the Microchip website (www.microchip.com).

The following is a discussion of the purpose and operation of each of the functions in Appendix A.

Main()

This simple function represents the main system master control program. This routine sets up the system master hardware (**init_hardware()**) and then calls the **sys_init()** routine.

Sys_Init()

This routine controls the detection and configuration of any circuit cards that are installed in the system. It performs a while loop that calls the **Assign_ID()** function, passing in the id value to be assigned, to determine if there are any unassigned devices on the bus. If the **Assign_ID()** function finds a device on the bus the

sys_init() routine then reads the memory size (address 0) and stores this in the global variable *addr* for later use. The variable *oe_bit* is then set and the device code is read (address 1). This value is used determine and execute the proper initialization/configuration code.

Note: the actual code has not been included in this example but, rather, comments are used to indicate where this code would go.

After the device configuration is finished a time stamp is written to the last three bytes of the array. Before returning to **main()** the value of the variable *id* is decremented so that a subsequent call to the **sys_init()** routine, which could be used to check for new devices that have been added to the system, will assign the next unused ID number.

Write_Byte()

This routine writes a single byte to the 24LCS61/62 device. The address that is to be written to must be placed in the global variable *addr* before calling this function in order for it work properly. The ID number of the device to be written to, and the data byte that is to be written are passed in by the calling function. The write is then initiated by issuing the write command, followed by the device ID, address byte and data byte.

Note: The state of the global variable *oe_bit* is logically OR'd, after being shifted left three bits, with the control code in order to provide control of the state of the $\overline{EDS}$ pin.

After the data byte is sent, a stop condition is generated to initiate the internal write cycle. The **Ack_Poll()** routine is then called to wait for the device to finish writing before continuing.

Read_Byte()

This routine reads a single byte from the 24LCS61/62 device. The device ID number and the address to be read are passed in by the calling function. The routine then sets the internal address pointer of the 24LCS61/62 by sending a write command (after ORing the *oe_bit* - see **Write_Byte()** above), followed by the device ID, and the address. A start condition is then generated followed by the read command (the *oe_bit* is again OR'd), and the device ID. The data byte is then read from the I^2C bus and returned to the calling function after generating a stop condition.

Ack_Poll()

This is a simple subroutine that is used after a write command is sent in order to allow the 24LCS61/62 to complete an internal write cycle before continuing. This is done by issuing repeated write commands to the 24LCS61/62 device, whose ID is passed in by the calling function, until an acknowledge is generated by the 24LCS61/62 indicating that it has completed the write cycle. It should be noted that the *oe_bit* is OR'd with the write command so that the state of the $\overline{EDS}$ pin is not inadvertently changed during the polling. After the acknowledge is received a stop condition is sent before the routine ends.

Clear_Addr()

This routine issues the Clear Address command to all 24LCS61/62 devices on the bus, which causes them all to reset their internal ID to 00, and to enter the unassigned state. The Assign ID command is followed by a dummy ID byte, and a stop condition.

Assign_ID()

Checks the bus for, and assigns an ID to, an unassigned 24LCS61/62 device. To do this the Assign ID command is sent, followed by the ID (which has been passed in by the calling function). The bus is then checked for an acknowledge being generated by an unassigned device. If no ACK is seen the function returns a '0' to indicate that no device was found. If an ACK has been generated, the function then reads the

six byte serial number, generates a stop condition which latches the assigned ID number into the 24LCS61/62, and returns a '1' indicating a device was found and assigned.

init_hardware()

This function initializes the hardware for the simplified system used to test and validate the operation of the code in this application note. This hardware uses a PIC17C4x with SCL and SDA on port pins RC6 and RC7 respectively. Both pins require an appropriate pullup resistor as detailed in the sw12c16.h file.

SUMMARY

This application note has detailed the basics of using the 24LCS61/62 in a multi board system. Several additions/enhancements could easily be added including implementing multi-byte page mode writes to the 24LCS61/62, detecting/handling I^2C bus errors, and creating the necessary code/interrupt to generate a periodic call from **main()** to **sys_init()** in order to detect and configure newly added devices. The source code provided was written in 'C", versus assembly language, for a couple of reasons. First, the use of 'C' makes the code more portable so that it can easily be implemented on any one of several microcontrollers. Second, it is believed that 'C' is a more 'readable' and therefore better serves as an instructional tool than does assembly language.

ADDITIONAL INFORMATION

24LCS61/62 Device Datasheet; Microchip Technology Inc.; DS21226

AN676: *"Physical Slot Identification Techniques for the 24LCS61/62"*; Microchip Technology, Inc.; DS00676

Please check Microchip's Worldwide Website at www.microchip.com for the latest version of source code.

APPENDIX A: SOURCE CODE

```
/*****************************************************************
*                                                                *
*  24LCS6x.c                                                     *
*  Source code for application note AN683 which demonstrates     *
*  using the Microchip Technology 24LCS61/62 software            *
*  addressable I2C Serial EEPROM.                                *
*                                                                *
*  06/xx/98    Original creation   R. Stoneking                  *
*              Created and compiled with MPLAB-C V1.21           *
*                                                                *
*  This code is written to run on a PIC17C4x device with SCL     *
*  and SDA on PORTC.6 and PORTC.7 respectively.  Both of these   *
*  pins require an external pullup resistor as explained in the  *
*  swi2c16.h file.                                               *
*                                                                *
*  This code does not make any attempt run the I2C bus at a      *
*  set speed, so the code will work independant of the           *
*  oscillator frequency (at faster speeds additional delays may  *
*  be required to prevent violating the I2C timing specs.        *
*                                                                *
*****************************************************************/
#include <17c42a.h>
#include <delays.h>
#include <math.h>
#include <swi2c16.h>          /* required header file from PICmicro */
                              /* library                            */
/******************* Function Prototypes***********************/
void     sys_init(void);
void     Clear_ID(void);
char     Assign_ID(char);
void     Write_Byte(char, char);
char     Read_Byte(char, char);
void     AckPoll(char);
void     init_hardware(void);
/*********** Global Defines*************************************/
#define  SET_WP_CMD           0x60
#define  READ_CMD             0x61
#define  WRITE_CMD            0x62
#define  ASN_ADD_CMD          0x64
#define  CLR_ADD_CMD          0x66
#define  FOUND                1
#define  NFOUND               0
/*********** Global Variables***********************************/
char     temp,
         id,
         oe_bit,
         rxbuf[16],
         addr,
         hour = 5,
         minute = 0x45,
         second = 0x33;

/*****************************************************************/
void main(void)
{
         init_hardware();     /* set up PICmicro */
         Clear_ID();
         sys_init();
         while(1);            * loop forever */
}        /* end main () */
```

```
/*****************************************************************
*  Name:   sys_init                                              *
*  Desc:   This routine controlls the querrying of the bus,      *
*          detection and assigning of new devices, and execution *
*          of any necessary configuration routines when a new    *
*          device is found.                                      *
*                                                                *
*  Inputs: None                                                  *
*                                                                *
*  Setup:  None                                                  *
*                                                                *
*  Return: Void                                                  *
*****************************************************************/
void sys_init(void)
{
     id = 1;                            /* start assigned id's at 1 */
     oe_bit = 0;                        /* EDS pin inactive */
     while(Assign_ID(id) == FOUND)
     {
     oe_bit = 1;                        /* set EDS pin low while
                                        addressing device */
     addr = Read_Byte(id,0x00);         /* read byte 0 for array size */
     temp = Read_Byte(id,0x01);         /* read byte 1 for device code */
     switch(temp)                       /* execute module dependant code */
     {
     case 0x01:                         /* Memory Module */
                                        /* add code here for Memory Module */
                                        break;

     case 0x02:                         /* I/O Module */
                                        /* add code here for I/O Module */
                                        break;

     case 0x03:                         /* Display Module */
                                        /* add code here for Display Module */
                                        break;

      default:
                                        /* add code here for unknown Module codes */
                                        break;
     }
     Write_Byte(id, second);            /* write time stamp to last 3
                                        bytes of array */
     addr--;
     Write_Byte(id, minute);
     addr--;
     oe_bit = 0;                        /* set EDS pin high during
                                        last command */
     Write_Byte(id, hour);
     id++;
     } /* end while() */
     id--;                              /* decrement id by one so next
                                        call to sys_init uses correct id */
}    /* end sys_init() */
/*****************************************************************
*  Name:   Write_Byte                                            *
*  Desc:   Write the data byte passed by the calling funtion to  *
*          the device specified in id, at the location pointed to*
*          by the global variable addr.                          *
*                                                                *
*  Inputs: id - the id of the device on the bus to be written to *
*          data_byte - byte to be written to the array           *
*                                                                *
*  Setup:  Global variable addr must be set before entry         *
*                                                                *
*  Return: Void                                                  *
*****************************************************************/
```

```
void     Write_Byte(char wr_id, char data_byte)
{
         RestartI2C();
         temp = WriteI2C(WRITE_CMD | (oe_bit << 3));
         AckI2C();
         temp = WriteI2C(wr_id);
         AckI2C();
         temp = WriteI2C(addr);              /* address byte */
         AckI2C();
         temp = WriteI2C(data_byte);         /* data byte */
         AckI2C();
         StopI2C();
         AckPoll(wr_id);                     /* wait for write cycle to finish */
}        /* end Write_Byte()                 */
/*****************************************************************
*  Name:   Read_Byte                                             *
*  Desc:   Reade the data from the device specified by id at the *
*          address specified by the paramater addr.              *
*                                                                *
*  Inputs: id - the id of the device on the bus to be written to *
*          addr - byte to be read in the array                   *
*                                                                *
*  Setup:  None                                                  *
*                                                                *
*  Return: Data byte read                                        *
*****************************************************************/
char Read_Byte(char rd_id, char loc)
{
         PORTD.0 = 1;
         StartI2C();
         temp = WriteI2C(WRITE_CMD | (oe_bit << 3));
         AckI2C();
         temp = WriteI2C(rd_id);
         AckI2C();
         temp = WriteI2C(loc);               /* address byte */
         AckI2C();
         NOP();
         RestartI2C();
         temp = WriteI2C(READ_CMD | (oe_bit << 3));
         AckI2C();
         temp = WriteI2C(rd_id);
         AckI2C();
         temp=ReadI2C();
         AckI2C();
         StopI2C();
         return I2C_BUFFER;
}        /* end Read_Byte()                  */
/*****************************************************************
*  Name:   AckPoll                                               *
*  Desc:   Send repeated Write commands and check for DUTk.      *
*          Return when ack received.  If no ack is received this *
*          will loop indefinately.                               *
*                                                                *
*  Inputs: Id of device to be polled                             *
*                                                                *
*  Setup:  None                                                  *
*                                                                *
*  Return: Void                                                  *
*****************************************************************/
void     AckPoll(char ack_id)
{
         do
    {
         RestartI2C();
         temp = WriteI2C(WRITE_CMD | (oe_bit << 3));
         AckI2C();
         temp = WriteI2C(ack_id);
```

```
         AckI2C();
      }while(BUS_STATUS.2);
      StopI2C();
}     /* end AckPoll()              */
/*****************************************************************
*  Name:   Clear_Addr()                                          *
*  Desc:   Send Clear Address Command to all devices on the bus, *
*          resetting all device ID registers to 00.              *
*                                                                *
*  Inputs: None                                                  *
*                                                                *
*  Setup:  None                                                  *
*                                                                *
*  Return: Void                                                  *
*****************************************************************/
void  Clear_ID()
{
      RestartI2C();
      temp = WriteI2C(CLR_ADD_CMD);
      AckI2C();
      temp = WriteI2C(0x00);
      AckI2C();
      StopI2C();
}
/*****************************************************************
*  Name:   Assign_ID                                             *
*  Desc:   Sends Assign ID command over the I2C bus and looks for*
*          the acknowlege from an unassigned device.  If a device*
*          is found, the six byte serial number is read into     *
*          the rxbuf[] array for use by the calling function.    *
*                                                                *
*  Inputs: Id to be assigned to device if detected               *
*                                                                *
*  Setup:  None                                                  *
*                                                                *
*  Return: 1 if device is found, 0 if no device found            *
*****************************************************************/
char  Assign_ID(char new_id)
{
      RestartI2C();
      temp = WriteI2C(ASN_ADD_CMD);
      AckI2C();
      temp = WriteI2C(new_id);
      AckI2C();
      if (BUS_STATUS.2)                       /* Check for unassigned device on bus */
      return NFOUND;                           /* and exit if none */
      temp = getsI2C(rxbuf,6);                /* Otherwise read SN */
      StopI2C();
      return FOUND;
}
void  init_hardware()
      {
      SPBRG = 0x0f;                           /* Use 0x07 for 19.2Kbps, 0x0f for 9600 bps     */
      TXSTA = 0x20;                           /* Async mode, 8 Data bits, txen = 1            */
      RCSTA = 0x90;                           /* 8 Data bits, spen = 1, cren = 1              */
      PORTB = 0x0;
      DDRB = 0xff;                            /* Port B all inputs                            */
      PORTC = 0xff;                           /* Set Port C to all ones                       */
      DDRC = 0xff;                            /* Port C all outputs for now                   */
      PORTD = 0;
      DDRD = 0x0;                              /* Port D all outputs                          */
      DDRE = 0xff;                            /* Port E all inputs                         */
      PIR = 0;                                /* Clear any pending interrupts              */
      CPUSTA.GLINTD = 1;                      /* Disable global interrupts                    */
      }                                       /*  end init_hardware()                         */

#include <swi2c16.lib>
#include <delays.lib>
```

MICROCHIP

AN690

I²C™ Memory Autodetect

Author: *Lucio Di Jasio*
Microchip Technology, Italy

INTRODUCTION

This application note describes a method to automatically detect the memory size of a serial EEPROM connected to an I²C bus. The topics include:

- Automatic detection of memory size on the I²C bus
- Standard I²C
- Smart Serial or the I²C Dilemma
- Another set of routines for I²C
- How to tell the addressing scheme
- How to tell the size
- Putting it all together
- Debugging
- Compatibility
- References

AUTOMATIC DETECTION OF MEMORY SIZE ON THE I²C BUS

The purpose of this application note is to show how to solve a common problem in microcontroller applications with Serial EEPROMs. User needs often dictate different memory sizes for different versions of an application, but cost constraints require the smallest possible memory to be used each time. A typical application example could be the base station (receiver) of a remotely controlled garage door opener. Versions capable of storing 4, 20, 200 or 1000 users could be implemented from a single source code complementing the controller with the appropriate memories.

Microchip currently offers a very broad range of memory capacities with I²C bus interface (from 16 bytes in the 24C00 up to 32k bytes in the 24C256).

The microcontroller has to be able to tell which memory it is dealing with on the I²C bus in order to address it properly.

There are two possible approaches to the problem, one is to provide some kind of configuration information to the controller by means of dip switches or jumpers, the other one is to make the controller capable of automatic detection. In this application note, we will show how to implement the automatic detection in an easy, safe and compatible way.

The software techniques explained in the following will be demonstrated on a generic mid-range PICmicro® microcontroller (MCU), PIC16C62A and can be tested immediately using a PICDEM2 demo board.

All the code can be adapted to any other PICmicro MCU (12, 14 and 16 bit core) and/or pin configuration with minor modifications to the source code.

Standard I²C

The I²C protocol utilizes a master/slave bi-directional communication bus. The master, usually a microcontroller that controls the bus, generates the serial clock (SCL) and originates the start and stop conditions. A Serial EEPROM is considered a slave device and is defined as a transmitter during read operations and generates acknowledges when receiving data from the master. The start and stop bits are utilized to control the bus. Normal operation begins with a start bit and ends with a stop bit. Following a start, commands begin with an 8 bit 'control' byte originated by the master. The control byte identifies the slave device to be addressed and defines the operation to take place. A typical control byte for a Serial EEPROM (slave address = `1010`) is shown in Figure 1. The control byte, therefore, consists of a start bit, a four-bit slave address, a read/write bit and an acknowledge. The slave address consists of the `1010` identifying address plus the three block or chip select bits A2,A1,A0.

Smart Serial or the I²C Dilemma [ref 3]

The I²C serial bus has many advantages over other common serial interfaces for serial embedded devices. The I²C bus with level-triggered inputs offers better noise immunity over edge-triggered technology. Opcodes are not needed to communicate with storage devices because all interfaces are intuitive and comparable to parallel devices.

But the standard protocol limits addressing up to a maximum of 16K bytes of memory on the bus via the 8-bit address and the three device or memory block select pins A0, A1, and A2 (8x2kbytes).

Herein lies the dilemma. With the advent of the more sophisticated personal communication devices such as cellular and full-featured phones, personal digital assistants and palm-top computers, 16K bytes is not enough!

So the Smart Serial concept grew from the industry's need for increased memory requirements in I²C embedded applications, smarter endurance performance, security needs, and the need for more functionality at lower power demands.

Microchip Technology has designed an addressing scheme for I²C Serial EEPROM based on the standard I²C protocol and device addresses, but incorporating an additional address byte for enabling the designer to use up to 256K bits per device and add from 1 to 8 devices on the system bus. This flexibility allows for future memory expansion and more advanced features in a smaller, more cost effective design.

For the first byte, or control byte, the Smart Serials adhere to the I²C protocol (reference Figure 2). The next 2 bytes (instead of one) define the address of the requested memory location.

Another Set of Routines for I²C bus

Many application notes have already been published by Microchip Technology on the I²C bus interface such as: AN515, AN537, AN558, AN567, AN608, AN554, AN578 and AN535. In the following, we will use techniques and code taken from those application notes as a base to build a new compact, powerful set of routines. The first step will be to modify a basic set of routines [ref1,2,4,6,8] to make them capable of producing Standard I²C and Smart Serial addressing, selecting the addressing scheme at run time by means of a flag (that we will call: SMART).

Listing 1 (`i2c.inc`) shows the new set of routines. As usual, there are two layers of functions:

- The lower layer (composed of routines: `BSTOP`, `BSTART`, `RXI2C`, `TXI2C`, `BITIN`, `BITOUT`, `ERR`; listing starts from line 153) deals with sending and detecting the single bits and bytes on the bus and contains no new code.
- The higher layer (composed of routines: `RDbyte`, `WRbyte` and `SETI2C`, from line 1 to 152) assembles commands and takes care of addressing schemes. This will be the focus of our discussion.

What is new here, is that we moved to function `SETI2C` (lines 112..152) all the code that deals with the details of the addressing scheme. This function gets a SMART flag as an input and provides Standard or Smart addressing according to its value. Both `RDbyte` and `WRbyte` rely on `SETI2C` for the command and address generation, and therefore are now compatible with Standard and Smart Serial.

Determining the Addressing Scheme

As a next small step toward automatic memory size detection we need to find a method to distinguish automatically between a Smart Serial and a Standard Serial EEPROM.

The algorithm proposed is very simple and compact, made up of only the following 4 steps:

1. Put in Smart Serial mode the I²C routines (set SMART flag).
2. Issue a write command to location 0000, writing a 1.

Note:	If the memory is a standard I²C, this command is interpreted as a sequential write command of two bytes that produces writing a 00 byte to location 0000 and a 01 byte to location 0001.

(0000) <- 00

(0001) <- 01

If the memory is a Smart Serial, then we get the correct interpretation.

(0000) <- 01

3. Put in Standard I²C Mode the I²C routines (clear the SMART flag).
4. Issue a read command of location 0000.

If the memory really is a Standard I²C, then this read command will give us the contents of location 0000, and that was set to 0!.

If the memory is a Smart Serial, we get a read command with a partial (incomplete) addressing.

What happens in this case is not really part of the I²C bus definition, so let's analyze two possible cases.

a) Partial addressing set only the most significant bits of the internal address register and leaves unattached the lower 8 bits. This means that we will read location 0000.

b) Partial addressing doesn't modify at all the address register. This means that the address remains equal to the last value set (by the last Smart Write) and reading gives the contents of location 0000.

If in both cases we end up reading a 1, that tells us that it was a Smart Serial memory. If a 0 was read, then it was a Standard I²C serial memory.

Listing 2) (`i2cauto.asm`) lines 108..120 implement in just 10 lines of assembly this simple algorithm.

Note:	Locations 0000 and 0001 are obviously corrupted through this procedure and there is no way to save and restore them (until the addressing scheme is known!).

Determining Memory Size

The last step toward automatic memory size detection is the development of an algorithm to tell the size of a memory given its addressing scheme. That is, suppose we know whether it is a Standard or Smart, we want to be able to measure its size.

We will base the detection algorithm on a simple assumption which is:

> If a memory is of size N, then trying to address locations out of the 0..N-1 range will produce a fall back in the same range (modulus N). Since the most significant (extra) address bits will be simply ignored, they are DON'T CARE bits to the device as can be easily verified from each device data sheet.

We can develop a simple test function to tell us whether a memory is of a given size N (or smaller).

In a high level pseudo language, such a test function could look like this:

EXAMPLE 1:

```
function TestIfSizeIs(Size N): boolean
(   // is memory range 0..N-1 ?
    var TEMP;
    TEMP = Read( 0000);

    if ( Read( N) == TEMP)
         Write( 0000, TEMP+1)

         if ( Read( N) == TEMP+1)
              Write( 0,TEMP-1)
              return( TRUE)
    // else
    return( FALSE)
) //end function
```

Having this function, we can then set up a loop to test memory sizes.

In the case of the Standard I^2C, we can loop and test from N=128 to N=2048 corresponding to models from 24C01 up to 24C16 doubling N at each iteration as in the following:

EXAMPLE 2:

```
function StandardI2CMemDetect() : integer
( // returns a model number 1..16

     N = 128
     MODEL = 1
     loop
          if (TestIfSizeIs( N))
              break
          else
              N=N*2
               MODEL=MODEL*2
     while(N<=2048)

return ( MODEL);
) //end function
```

Similarly, a function to measure Smart Serial memories will loop with N=4096 up to N=32768.

Please note that in this second algorithm, no memory location had to be reserved. Even location 0 that is modified could always be saved and restored by the test algorithms.

PUTTING IT ALL TOGETHER

Now all the pieces of the puzzle are ready and we can complete our automatic memory size detection routine. First we determine the addressing scheme, and once that is known, we enter a loop to measure the actual memory size. Depending on the addressing scheme, we will enter the loop with different initial values corresponding to the different ranges of memory according to the memory models available on the market.

Listing 2 (`i2cauto.asm`) lines 136..174 implement in assembly in a very compact way both algorithms.

Debugging

Assembling the code and testing it on a PIC16C62A on a PICDEM2 board or any other target board (after modifying the pin definitions in listing 2 (`i2cauto.asm`) lines 48..60) will prove the functionality of the proposed code. Just insert an I^2C memory in the DIL socket on the PICDEM2 board, power up or press the reset button, and voila', on the LEDs will appear the binary representation of the memory TYPE value according to Table 1.

TABLE 1: MEMORY TYPE VALUE

Standard I^2C			Smart Serial		
Type	**Size**	**Model**	**Type**	**Size**	**Model**
01	128	24C01/21/41	32	4096	24C32
02	256	24C02/62	64	8192	24C65/64
04	512	24C04	128	16384	24C128
08	1024	24C08	0	32768	24C256
16	2048	24C16/164			

The reader is invited to experiment and modify further this software to adapt it to their specific needs. When doing so, we strongly recommend having at hand the SEEVAL kit, a cheap and effective tool from Microchip Technology that allows the designer to read/write any Serial EEPROM and connects to any PC through the serial port. Further consider the "Endurance" software tool from Microchip Technology, while designing memory applications where reliability and endurance are critical.[ref 9,10]

Compatibility

While most of the code presented strictly follows the existing I^2C and Smart Serial standards, it should be compatible with any Serial EEPROM device from any manufacturer, that adheres to such standards. Only Microchip Serial EEPROMs were tested. It is left up to the user to validate this code for Serial E^2 from other manufacturers.

Further, there is some space for discussion, as a possible future compatibility issue, on the addressing scheme detection method. As a matter of fact, the behavior of the serial memory in case of partial addressing (as it occurs during step 4 in the case of Smart Serial) is not part of the specification. While it works with current implementations of the Smart Serial protocol (from Microchip and up to the 24C256), it is not guaranteed to do so in the future.

References

[1] AN515 Communicating with I^2C™ Bus Using the PIC16C5X, Bruce Negley

[2] AN535 Logic Powered Serial EEPROMs, R. J. Fisher and Bruce Negley

[3] AN558 Using the 24xx65 and the 24xx32 with Stand-alone PIC16C54 Code, Dick Fisher and Bruce Negley

[4] AN567 Interfacing the 24LCxxB Serial EEPROMs to the PIC16C54, Bruce Negley

[5] AN608 Converting to 24LCXXB and 93LCxx Serial EEPROMs, Nathan John

[6] AN536 Basic Serial EEPROM Operation, Steve Drehobl

[7] AN554 Software Implementation of I^2C™ Bus Master, Amar Palacherla

[8] AN559 Optimizing Serial Bus Operations with Proper Write Cycle Times, Lenny French

[9] AN537 Serial EEPROM Endurance, Steve Drehobl

[10] AN602 How to get 10 Million Cycles Out of Your Microchip Serial EEPROM, David Wilkie

FIGURE 1: CONTROL BYTE ALLOCATION

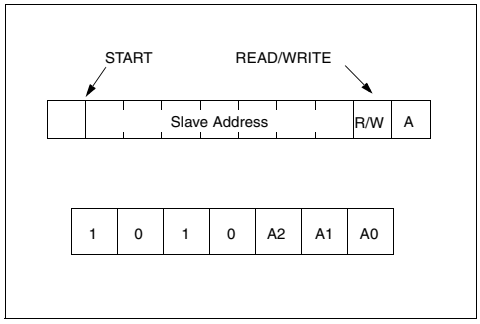
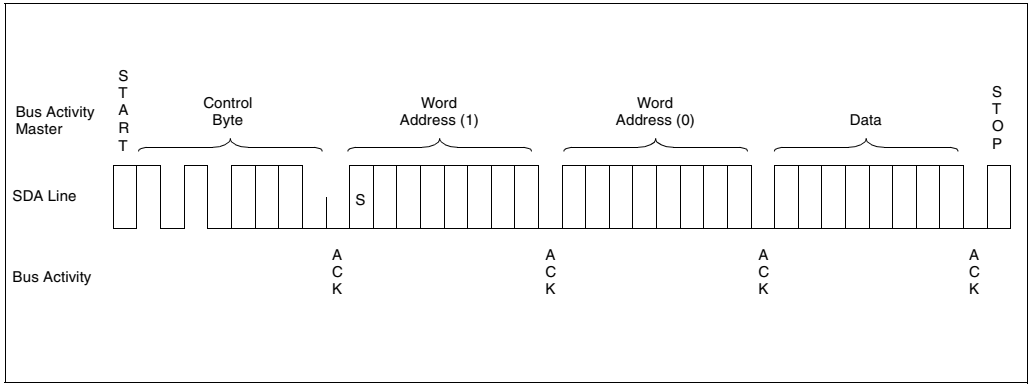


FIGURE 2: BYTE WRITE



Please check Microchip's Worldwide Website at www.microchip.com for the latest version of source code.

APPENDIX A: SOURCE CODE

LISTING 1: I2C.INC

```
;***********************************************************************
;*  Filename:   I2C.INC
;***********************************************************************
;*  Author:     Lucio Di Jasio
;*  Company:    Microchip Technology
;*  Revision:   RevA0
;*  Date:       5-7-98
;*  Assembled using MPASM v02.15
;***********************************************************************
;*  Two wire/I2C Bus READ/WRITE Sample Routines
;*  both Smart Serial and Standard I2C addressing schemes supported
;*  PIC16CXXX mid-range (14 bit core) version
;*
;*  Note:  1) All timing is based on a reference crystal frequency of 4MHz
;*            which is equivalent to an instruction cycle time of 1 usec.
;*         2) Address and literal values are read in hexidecimal unless
;*            otherwise specified.
;*************************************************************************
;*
;*       Register File Assignment
;*************************************************************************
    CBLOCK
        FLAGS
        INDHI       ; address
        INDLO
        DATO        ; data buffer for read write functions
        ERCODE      ; error code (see table below)
        EEBUF       ; read write buffer
        SLAVEbuf    ; SLAVE address (+ addrHi on 24LC16)
        COUNT
        AUX
    ENDC
;***********************************************************************
; flag definitions
;
#define     FLAG_EE     FLAGS,0     ; I2C bus error
#define     SMART       FLAGS,1     ; Smart(1) Standard(0)
;
;*************************************************************************
;*                     Bit Assignments
;*************************************************************************

#define     SLAVE   B'10100000' ; Device address (1010xxx0)

; error codes
#define     ERR_NACK    1       ; no ACK reading
#define     ERR_STOP    2       ; SDA locked in STOP
#define     ERR_TOWR    3       ; time out in read (>20ms)
#define     ERR_LOCK    4       ; SDA locked in BITOUT

;*************************************************************************
;*      RDbyte
;*      read one byte from serial EEPROM device
;*
;*      Input   :       INDHI/LO
;*                      SLAVE   = device address (1010xxx0)
;*      Output  :       DATO    = data read from serial EEPROM
;*************************************************************************
;
```

```
RDbyte  bcf     FLAG_EE         ; reset error flag
        call    SETI2C          ; set address pointer

; enter here for sequential reading

RDnext  call    BSTART          ; START
    movf    SLAVEbuf,W      ; use SLAVE addr(+IndHi se 24LC16)
    movwf   EEBUF
    bsf     EEBUF,0         ; it's a read command
    call    TXI2C           ; Output SLAVE + address + read command
    call    RXI2C           ; read in DATO and ACKnowledge
    movf    EEBUF,W
    movwf   DATO

    bsf     STATUS,C        ; set ACK = 1 (NOT ACK)
    call    BITOUT          ; to STOP further input
    goto    BSTOP           ; generate STOP bit

;*****************************************************************************
;*      WRbyte
;*      write one byte to EEPROM device
;*
;*      Input   :       DATO    = data to be written
;*                      INDHI/LO= EEPROM data address
;*                      SLAVE   = device address (1010xxx0)
;*                      PROT    = 1-> SmartSerial | 0> Standard
;*      Output  :       FLAG_EE = set if operation failed
;*****************************************************************************

WRbyte  bcf     FLAG_EE         ; reset error condition
    call    SETI2C          ; set address pointer
    movf    DATO,W          ; move DATO
    movwf   EEBUF           ; into buffer
    call    TXI2C           ; output DATO and detect ACKnowledge
    call    BSTOP           ; generate STOP bit

; loop waiting for writing complete
        movlw   .80             ; 80 test=20ms timeout
        movwf   AUX
WRpoll  CLRWDT                  ; keep the WDT from resetting
        bcf     FLAG_EE
        call    BSTART           ; invia start
        movlw   SLAVE
        movwf   EEBUF
        call    TXI2C           ; ed un comando di scrittura
        btfss   FLAG_EE         ; se non da ACK -> ercode 3 -> BUSY
        goto    WRpollE
WRbusy  decfsz  AUX,F
        goto    WRpoll
        movlw   ERR_TOWR        ; time out in scrittura
        call    ERR
WRpollE goto    BSTOP           ; exit sending the stop condition

;*****************************************************************************
;*      SETI2C
;*      set the address pointer at INDHI/LO, use Smart or Standard
;*      addressing scheme according to SMART flag
;*
;*      Input   :       INDHI   = EEPROM data address
;*                      INDLO
;*                      SLAVE   = device address (1010xxx0)
;*                      SMART   = 1-> Smart Serial  | 0>  Standard I2C
;*      Output  :       SLAVEbuf    for sequential read
;*****************************************************************************
SETI2C
        btfsc   SMART           ; if clear -> Standard I2C
```

```
        goto    Smart           ; if set   -> Smart Serial

Standard
        bcf     STATUS,C        ;
        rlf     INDHI,W         ; add address MSb
        iorlw   SLAVE           ; to slave address
        movwf   EEBUF
        movwf   SLAVEbuf        ; save for sequential read
        call    BSTART          ; generate START bit
        call    TXI2C           ; output first comand byte
        goto    SETseq

Smart
        movlw   SLAVE           ; prepare slave address
        movwf   EEBUF
        movwf   SLAVEbuf        ; save for sequential read
        call    BSTART          ; generate START bit
        call    TXI2C           ; output first command byte
        movf    INDHI,W         ;
        movwf   EEBUF           ; output address MSB
        call    TXI2C

SETseq
        movf    INDLO,W         ; send address LSB
        movwf   EEBUF
        goto    TXI2C           ; Output WORD address

;*****************************************************************************
;*      TXI2C
;*      transmit 8 data bits
;*
;*      Input   :       EEBUF
;*      Output  :       none
;*****************************************************************************
TXI2C
        movlw   .8        ; Set counter for eight bits
        movwf   COUNT

TXlp
        rlf     EEBUF,F         ; data bit in CARRY
        call    BITOUT          ; Send bit
        decfsz  COUNT,F         ; 8 bits done?
        goto    TXlp            ; No.

        call    BITIN           ; Read acknowledge bit
        movlw   ERR_NACK
        btfsc   STATUS,C        ; Check for acknowledgement
        call    ERR             ; No acknowledge from device
        return

;*****************************************************************************
;*      BITOUT
;*      send single bit
;*
;*      Input   :       bit in CARRY
;*      Output  :       Bit transmitted over I2C
;*     Error bits set as necessary
;*****************************************************************************
BITOUT
        btfss   STATUS,C        ; is it 0/1?
        goto    Bit0

Bit1
        bsf     STATUS,RP0      ; select RAM bank 1
        bsf     SDA             ; input SDA (pull up->1)
        bcf     STATUS,RP0      ; back to RAM bank 0
```

```
        movlw   ERR_LOCK
        btfss   SDA             ; Check for error
        call    ERR              ; SDA locked low by device
        goto    Clk1

Bit0
        bsf     STATUS,RP0      ; select RAM bank 1
        bcf     SDA              ; Output SDA
        bcf     STATUS,RP0      ; back to RAM bank 0
        bcf     SDA              ; clear 0
        nop                      ; Delay

Clk1
        bsf     SCL             ; rise SCL
        nop
        nop
        nop                      ; Timing delay 4us minimum
        nop
        nop
        bcf     SCL              ; lower SCL
        return

;
;*****************************************************************************
;*      RXI2C
;*      receive eight data bits
;*
;*      Input   :       None
;*      Output  :       RXBUF = 8-bit data received
;*****************************************************************************
RXI2C
        movlw   .8               ; 8 bits of data
        movwf   COUNT
        clrf    EEBUF

RXlp
        call    BITIN            ; new bit in CARRY
        rlf     EEBUF,F          ; enter new bit
        decfsz  COUNT,F          ; 8 bits?
        goto    RXlp
        return

;
;*****************************************************************************
;*      BITIN
;*      Single bit receive
;*
;*      Input   :       None
;*      Output  :       EEBUF,0 bit received
;*****************************************************************************
BITIN
        bsf     STATUS,RP0      ; select RAM bank 1
        bsf     SDA              ; Set SDA for input
        bcf     STATUS,RP0      ; back to RAM bank 0
        bsf     SCL              ; Clock high
        nop
        nop
        nop
        nop                      ; provide minimum Tset up
        CLRC
        btfsc   SDA              ; Read SDA pin in CARRY
        bsf     STATUS,C
        bcf     SCL              ; Return SCL to low
        return

;*****************************************************************************
```

```
;*      START bit generation
;*
;*      input   : none
;*      output  : initialize bus communication
;*************************************************************************
BSTART
        bsf     STATUS,RP0      ; select RAM bank 1
        bsf     SDA             ; SDA input (pull-up ->1)
        bcf     STATUS,RP0      ; back to RAM bank 0
        bsf     SCL             ; Set clock high
        nop
        nop
        nop
        nop                     ; 5us before falling SDA
        bsf     STATUS,RP0      ; select RAM bank 1
        bcf     SDA             ; SDA output
        bcf     STATUS,RP0      ; back to RAM bank 0
        bcf     SDA             ; set SDA = 0
        nop
        nop
        nop
        nop                     ; 4us before falling SCL
        bcf     SCL             ; Start clock train
        return

;*************************************************************************
;*      STOP bit generation
;*
;*      Input   :       None
;*      Output  :       Bus communication, STOP condition
;*************************************************************************
BSTOP
        bsf     STATUS,RP0      ; select RAM bank 1
        bcf     SDA             ; SDA output
        bcf     STATUS,RP0      ; back to RAM bank 0
        bcf     SDA             ; set SDA = 0
        bsf     SCL             ; Set SCL high
        nop
        nop
        nop
        nop                     ; 4us before rising SDA
        bsf     STATUS,RP0      ; select RAM bank 1
        bsf     SDA             ; SDA input (pull-up ->1) while SCL high
        bcf     STATUS,RP0      ; back to RAM bank 0
        movlw   ERR_STOP        ; Ready error code
        btfss   SDA             ; High?
        call    ERR             ; Error, SDA locked before STOP

        bcf     SCL             ; lower SCL
        return
;
;*************************************************************************
;*      Two wire/I2C - CPU communication error status table
;*
;*      input   :    W-reg   = error code
;*      output  :    ERCODE  = error code
;*                   FLAG(ERROR) = 1
;*************************************************************************
ERR
        bcf     STATUS,RP0      ; back to RAM bank 0
; record last error
        movwf   ERCODE          ; Save error code
        bsf     FLAG_EE         ; Set error flag
        return
```

LISTING 2: I2CAUTO.ASM

```
    LIST    n=0, c=132
    RADIX   HEX
    PROCESSORPIC16C62A
;*****************************************************************************
;*  Filename:   I2CAUTO.ASM
;*****************************************************************************
;*  Author:     Lucio Di Jasio
;*  Company:    Microchip Technology
;*  Revision:   RevA0
;*  Date:       5-7-98
;*  Assembled using MPASM v02.15
;*****************************************************************************
;*  Include files:
;*      p16c62A.inc rev1.01
;*
;*****************************************************************************
;*  software detection of I2C memory size
;*
;*      PIC16CXXX                /+5V
;*    +-----------+              |
;*    |        Vdd+--------------+--------+    24CXXX
;*    |           |              +++       |  +--------+
;*    |           |              | |       +--+Vdd     |
;*    |           |              | | 4k7     |         |
;*    |           |              +++           |         |
;*    |        RC4+--------------+-----------+SDA      |
;*    |        RC3+--------------------------+SCL      |
;*    |           |                          |         |
;*    |        Vss+--------------+-----------+Vss      |
;*    +-----------+              |           +--------+
;*                               GND
;*
;* can be tested on a PICDEM2 demo board
;*****************************************************************************

    INCLUDE     "P16C62A.INC"

    __CONFIG    _XT_OSC & _CP_OFF & _WDT_ON
    __IDLOCS    H'62A0'

;*****************************************************************************
;*  external 4MHZ crystal oscillator
;*  no code protection
;*  no watchdog
;*  ID code is "62A0"
;*****************************************************************************

;   pin assignments

#define     SDA     PORTC,4     ; i I2C SDA
#define     SCL     PORTC,3     ; o I2C SCL

MASKA       equ     0FF         ; unused all inputs

MASKB      equ     00          ; all outputs to LEDs

MASKC       equ     b'11110111' ; SCL and SDA on this port
; enable SCL as output
;
;-------------------------------------------------------------
;   RAM assignments
;
    CBLOCK      20
        TEMP
        SIZELO      ; memory size
```

```
        SIZEHI
        TYPE        ; memory type
    ENDC

;*****************************************************************

            org     00              ; reset vector

            goto    Start

;*****************************************************************

            org     04              ; interrupt vector

            retfie                  ; esce riabilitando gli interrupt

;*****************************************************************

    INCLUDE "i2c.inc"

;*****************************************************************
;*  MemDetect,
;*  automatic detection of memory size
;*
;*  INPUT:
;*      none
;*  OUTPUT:
;*      SIZEHI/LO   memory size as detected
;*      TYPE        memory type (see table below)
;*      FLAG_EE     bus error flag
;*      ERCODE      bus error code
;*
;*          Standard I2C                Smart Serial
;*     TYPE   SIZE  MODEL           TYPE    SIZE    MODEL
;*      01     128  24C01/21/41       32    4096    24C32
;*      02     256  24C02/62          64    8192    24C65/64
;*      04     512  24C04            128 - 16384    24C128
;*      08    1024  24C08              0 - 32768    24C256
;*      16    2048  24C16/164
;*
;*****************************************************************
MemDetect
            clrf    INDHI       ; address 0000h
            clrf    INDLO
            bsf     SMART       ; write(smart, 0000, 1)
            movlw   1
            movwf   DATO
            call    WRbyte
            bcf     SMART
            call    RDbyte      ; read(standard, 0000)
            movf    DATO,W
            btfsc   STATUS,Z
            goto    StandardD
SmartD
            bsf     SMART       ; it is a Smart Serial
            movlw   HIGH(.4096)
            movwf   SIZEHI      ; size = 4096 byte
            clrf    SIZELO
            movlw   .32
            movwf   TYPE        ; start with TYPE = 24C32
            goto    TestD


StandardD
            bcf     SMART       ; it is a Standard Serial
            movlw   .128
            movwf   SIZELO      ; size = 128 byte
            clrf    SIZEHI
```

```
            movlw   01
            movwf   TYPE        ; start with TYPE = 24C01

TestD
            call    RDbyte      ; TEMP=read(0)
            movf    DATO,W
            movwf   TEMP
LoopDet     movf    SIZELO,W    ; DATO=read(SMART, size)
            movwf   INDLO
            movf    SIZEHI,W
            movwf   INDHI
            call    RDbyte
            movf    DATO,W
            xorwf   TEMP,W      ; compare TEMP with DATO
            btfss   STATUS,Z
            goto    LoopDN
            incf    TEMP,W      ; if same value than TEMP=TEMP+1
            movwf   TEMP
            movwf   DATO
            clrf    INDHI
            clrf    INDLO
            call    WRbyte      ; write(SMART, 0000, TEMP)
            movf    SIZELO,W    ; if (read(SMART, size) == TEMP)
            movwf   INDLO
            movf    SIZEHI,W
            movwf   INDHI
            call    RDbyte
            movf    DATO,w      ; if still same value it means
            xorwf   TEMP,W      ; we reached the actual memory size
            btfsc   STATUS,Z
            goto    DetEx

LoopDN
            bcf     STATUS,C    ; double memory size
            rlf     SIZELO,F
            rlf     SIZEHI,F
            bcf     STATUS,C
            rlf     TYPE,F      ; double TYPE code
            btfss   TYPE,4
            goto    LoopDet     ;
DetEx
            nop
            return

;*****************************************************************
; init ports and option register
;
Start
            bsf     STATUS,RP0      ; select RAM bank 1
            movlw   MASKA           ; set tris registers
            movwf   PORTA           ; PORTA
            movlw   MASKB           ;
            movwf   PORTB           ; PORTB
        movlw   MASKC           ;
            movwf   PORTC           ; PORTC

            movlw   b'00000111'     ; enable pull_ups, prescale TMR0 1:256
            movwf   OPTION_REG

            bcf     STATUS,RP0
            clrf    FLAGS           ; reset all flags
```

```
;*****************************************************************
;

Main
            call    MemDetect       ; determine memory size

            movf    TYPE,W          ; if using a PICDEM2 board
            movwf   PORTB           ; send TYPE to the LEDs

MainLoop
            goto    MainLoop        ; stuck in the loop until reset

        END
```

AN698

Microchip 93 Series Serial EEPROM Compatibility

Author: *Rick Stoneking*
Microchip Technology Inc.

INTRODUCTION

With Microchip Technology's introduction of the 93XX46, 93XX56, and 93XX66 devices in the 'A' and 'B' versions, a certain number of compatibility questions have arisen. This application note is intended to discuss, in detail, the differences between each of the variations of the 93XX series of devices offered and address all of the technical considerations related to the conversion from one device to another.

There are several key areas in which compatibility, particularly with respect to conversion from the 'original' 93XX46/56/66 and 93XX46B/56B/66B to the new 93XX46A/B, 93XX56A/B, and 93XX66A/B, must be specifically addressed. These areas include the following:

- Operating Voltage Range
- Write Cycle Initiation
- CS Functionality
- Ready/$\overline{\text{Busy}}$ Polling
- Tying DI and DO together
- Noise Immunity

Each of these points is addressed individually in this app note under a section heading of the same name.

PRODUCT FAMILY OVERVIEW

The current 93 series product offering from Microchip Technology totals 31 devices when including both the original and the new devices. In addition, each device is offered in one or more temperature grades. This fairly large number of individual products is mitigated, from a compatibility standpoint, by the fact that for many of the devices a single die is used for multiple 'part numbers'. For example, the original 93AA46, 93AA46B, 93LC46, and 93LC46B are all 'made' from the exact same die. The 'AA' and 'LC' versions are separated by testing to different V_{DD} limits and the 'B' version is a bonding option. In other cases multiple parts are created from a single die through the use of metal level options during fabrication. The significance of this is in the fact that in any case where a single die makes multiple parts (as in the example of the 93XX46/46B above) the functionality is identical for all variations. This effectively reduces the number of different devices that have to be addressed in this application note by allowing several devices to be grouped together for the purpose of functional discussions. Further reduction is possible by grouping devices of different array sizes together when their functionality is otherwise identical. Finally, devices can also be grouped together regardless of whether they utilize X8 or X16 communication.

Table 1 provides a list of devices that have been grouped together such that the members of each group are functionally identical with the exceptions of voltage range, array size, and/or X8/X16 communication (which are irrelevant for the purposes of this application note)

TABLE 1: FUNCTIONAL GROUPING

Group	Devices
1	93C06, 93C46
2 (Original)	93AA46/46B, 93LC46/46B, 93AA56/56B, 93LC56/56B, 93C56/56B, 93AA66/66B, 93LC66/66B, 93C66/66B
3	93AA76, 93LC76, 93C76 93AA86, 93LC86, 93C86
4 (New)	93LC46A/46B, 93LC56A/56B, 93LC66A/66B
5	93C46B, 93C56A/56B, 93C66A/66B

For the remainder of this document reference will only be made to the functional group number listed above rather than to specific devices. In all cases any device from within the group will function as described.

OPERATING VOLTAGE RANGE

One key difference between Group 2 'LC' devices and Group 4 'LC' devices is the range of V_{DD} over which the devices operate. Group 2 devices are specified to operate from 2.0V to 6.0V and Group 4 devices are specified from 2.5V to 6.0V. Group 4 devices have a V_{DD} threshold detect that is set at ~2.2V (nominal) which prevents operation when V_{DD} drops below that level. In any application where a Group 2 'LC' device is being used at a V_{DD} level below 2.5V, the equivalent 'AA' Group 2 device should be used. Because this conversion does not require using a device from a new functional group (see Table 1) there are no other compatibility issues. The second compatibility issue arises from converting from Group 2 'C' devices to Group 5 'C' devices. Group 2 devices, while specified as 'C' which indicates a V_{DD} operating range of 4.5V to 5.5V, will actually operate down to ~1.8V due to the fact that the V_{CC} threshold detection circuit has a ~1.5V trip point. Group 5 'C' devices, however, have a V_{CC} thresh-

old detect trip point of ~3.8V. Because of this difference in trip point settings, applications that previously used Group 2 'C' devices, and have a requirement to operate below 4.5V, should be converted to Group 4 'LC' devices, which will operate down to 2.5V.

WRITE CYCLE INITIATION

For the 93 Series EEPROM's, the internal write cycle is initiated by one of two events depending upon the functional group. For Groups 1, 3, and 5 the write cycle is initiated by the rising edge of CLK for the last data bit (D_0). For Groups 2 and 4, the write cycle is not initiated until CS is lowered.

The devices in Groups 1 and 5 are targeted specifically at 5V, extended temperature applications. Group 5 devices were intentionally designed to have the same write cycle CS functionality as Group 1 devices in order to serve as drop in replacements. Group 3 devices are unique in that they span the entire voltage range but also initiate the write cycle on the rising edge of CLK.

A potential compatibility issue regarding write cycle CS functionality only occurs if converting from Group 1, 3 or 5 devices to Group 2 or 4 devices. The issue is that with Groups 1, 3 or 5 devices the write cycle is initiated as soon as the rising edge of CLK is detected for the last data bit and the application firmware is not required to lower the CS line until: either the next command is to be sent (Group 1 or, 2) not at all (Group 3 and 5). So if a Group 2 or 4 device were to be placed into such an application the write cycle would not be initiated until just before the next command or not at all. In both cases the command following the write would be ignored. For case one, the part would be in a write cycle and would not recognize the command, and for case two, the device would see an invalid command and abort (no falling edge of CS after write command).

When converting from Group 2 or 4 devices to Group 1, 3 or 5 devices there are no compatibility issues related to write cycle CS functionality.

CS FUNCTIONALITY

In addition to the write cycle initiation compatibility question discussed in the previous section there are a couple of other conditions related to CS functionality that must be understood. These are related to both how the CS function works for the various devices, as well as to what conditions may occur, due to the way the firmware is written, which may cause problems. Table 2 details the differences in the way the CS pin functions for the various device groups. As can bee seen from the table (Item 1) devices in Groups 2 and 4 require that CS be brought low, after the write command is given, in order for the write cycle to be initiated.

This presents a compatibility problem only when converting from Group 3 or 5 devices to Group 2 or 4 (i.e. reducing EEPROM array size or moving from a 5V design to 2.5V or 1.8V design). In order to minimize this compatibility problem, CS should always be brought low after sending a write command, regardless of which device is being used.

Item 2 in Table 2 shows that Group 1 devices require a low to high transition on the CS pin after the write cycle has completed if data polling was used. If the CS line is not brought low after the write command is sent (i.e. data polling is not used), then no transition is required on CS after the write cycle completes (Item 3). To avoid this compatibility problem, CS should always be brought high at the end of a data polling routine.

Items 4 and 5 show that Group 1 devices require a low to high transition to occur after a write cycle completes in order for the next command to be recognized.

TABLE 2: CS FUNCTIONALITY

Item	Group 1	Group 2	Group 3	Group 4	Group 5	Description
1	No	Yes	No	Yes	No	Does CS have to go low to start the write Cycle?
2	Yes	No	No	No	No	Does CS, if data polling is used, have to have a low to high transition after the write cycle in order for the next command to be recognized?
3	Yes	N/A	N/A	Yes	Yes	If CS is not lowered after a write command, and is held high through the write cycle, will the next command be recognized?
4	No	Yes	Yes	Yes	Yes	If CS is brought high during write cycle will the next command be recognized?
5	No	Yes	Yes	Yes	Yes	If CS changes during write cycle, and is high at the end of the write cycle, will the next command be recognized?

READY/$\overline{\text{BUSY}}$ POLLING

Ready/$\overline{\text{Busy}}$ polling is a method that allows the MCU to poll the serial EEPROM device during a write cycle to determine when the write cycle has completed. The basic operation is that once a write cycle has been initiated the serial EEPROM will drive the DO line low, if CS is brought high during the write cycle, until the write cycle is complete. Once the write cycle has completed the DO pin is driven high.

There are two methods of functional operation of the DO pin for ready busy polling. The first (or original) method applies to Group 1, 2, and 3 devices. For these devices when CS is brought high during a write cycle, the DO pin will drive low while the write is in progress and then drive high once complete. If the CS pin is brought high at any time after a write cycle has completed, the DO pin remains in a high impedance state.

For Group 4 and 5 devices the operation is the same except that when CS is brought high after a write cycle is complete, the DO pin will drive high to indicate a ready state. This was done because on the original devices it was possible to miss the Ready signal if data polling was not started until after the write cycle completed. The DO pin will continue to drive the ready signal out until a start condition is detected, at which time the DO pin will return to a high impedance state. This difference in operation only becomes a compatibility issue if the DI and DO lines are being tied together for two wire operation (see the next section for a separate discussion of this issue).

TYING DI AND DO TOGETHER

When using the new 93 series devices (Group 4 and 5) with DI and DO tied directly together a bus contention issue will arise and in most cases will render the bus inoperable. The problem is a result of the change that was made to the operation of the DO pin during Ready/$\overline{\text{Busy}}$ polling. The original 93 series devices (Group 1,2 and 3) all tri-state the DO pin any time CS is brought high after a write cycle has finished. The new 93 series devices (Group 4 and 5) do not tri-state DO in the same situation, but instead drive the DO pin high to indicate the ready state. Because the DO pin (and therefore the DI pin also when DO and DI are tied together) is being actively driven high by the serial EEPROM a potential for bus contention exists if the MCU attempts to drive the DI line low. The actual voltage level of the bus will be a function of the sink capability of the MCU I/O pin and the source capability of the serial EEPROM. To prevent this bus contention issue it is necessary to use a resistor (~10K Ohm typical) to tie DI to DO instead of a direct connection.

Another potential problem with tying DI and DO occurs if CS is brought high after the write cycle has been initiated and valid clock transitions occur on the CLK line during the write cycle. This can cause the device to see an inadvertent START condition when the write cycle ends (and the DO line drives high to the Ready state). This is caused by the fact that the Twc of the device is variable (depending upon process/Vcc/temperature variations) and, therefore, causes the write cycle to end in an asynchronous fashion with respect to the clock. For example, if CS is high when the write cycle ends and DO goes high (thereby pulling DI high), then a low to high transition on the CLK will be seen as a valid START condition when it may not have been intended to be a START condition.

NOISE IMMUNITY

Probably the single biggest compatibility issue (related to conversions) exists between the original 93 series devices and the new devices due to noise immunity. The new devices (Group 4 and 5) are much faster internally than the older devices which makes them more susceptible to noise. This noise susceptibility becomes an acute problem during power up/down, during which time the state of the CS pin may be allowed to float up by an MCU that is not fully powered up. If this occurs it is very possible that noise on the CLK and DI pins can occur in the correct sequence required to issue an EWEN command followed by a spurious erase or write command. The most common occurrence is that of the EWEN followed by an ERASE command with the address and data bits all set to '1', so the last byte of the array is erased to 'FF'. The solution for this problem is a pull down resistor on the CS pin, which will prevent the device from responding to the spurious 'commands' on the CLK and DI lines.

DEVELOPING ROBUST CODE AND HARDWARE

In order to design a system (both hardware and firmware) that will be portable when changing the particular 93 series device being interfaced with, the following suggestions should be considered:

- The CS pin should be brought low at the end of every command issued to the device.
- Utilize Ready/$\overline{\text{Busy}}$ polling to verify that the device has completed the write operation rather than a delay loop of fixed duration.
- Create a data polling routine that leaves the CS line low when it is finished.
- Do not allow transitions on the CLK line (if CS is high) while the device is in a write cycle.
- Utilize a resistor between DO and DI if implementing a two wire interface.
- Utilize a pull down resistor on the CS line to prevent spurious commands during power up/down.

SUMMARY

When converting from one 93 series device to another there are a few key items that must be examined from the firmware side, as well as from the hardware side. By careful attention to these items it is possible to simplify and expedite the conversion process, as well as develop robust code and designs that will minimize the requirement for future changes.

NOTES:

MICROCHIP

AN709

System Level Design Considerations When Using I²C™ Serial EEPROM Devices

Author: *Rick Stoneking*
Microchip Technology Inc.

INTRODUCTION

Developing systems that implement the I²C protocol for communicating with serial EEPROM devices requires that a certain key factors be considered during the hardware and software development phase if the system is to achieve maximum compatibility and robustness. This application note discusses these factors, both hardware and software, to help insure that an optimal system design is achieved. This application note is limited to single master systems and therefore does not specifically address the unique requirements of a multi-master system. However, the concepts presented in this application note apply equally as well to those systems.

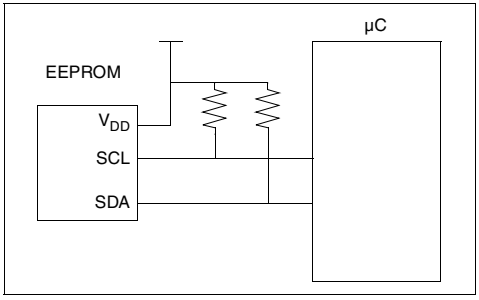
CONDITIONS TO BE CONSIDERED

Due to the bi-directional nature of the data bus devices operate in both transmit and receive modes at various times. In order to make this bi-directional operation possible the protocol must define specific times at which any given device may transmit or receive, as well as define specific points in the protocol where the functions are swapped (i.e. the transmitter becomes the receiver and the receiver becomes the transmitter). There are a number of events which could potentially cause this sender/receiver 'synchronization' to be lost, which can result in situations where:

- Both the master and the slave are in a send mode.
- Both the master and the slave are in a receive mode.
- The 'bit count' is off by one or more bits between the master and the slave.

These events, which include the microcontroller being reset during I²C communication, brown-out conditions, excessive noise on the clock or data lines, and improper bus input levels during power up, can be effectively neutralized through a combination of hardware and software techniques.

FIGURE 1: RECOMMENDED HARDWARE CONFIGURATION



INSURING 'BUS-FREE' DURING POWER-UP

In order to insure that the internal state machine of the serial EEPROM is correctly initialized at power up, it is crucial to guarantee that the device sees a 'bus-free' condition (defined as both SCL and SDA being high) until $V_{DD_{min}}$ has been reached. The ideal way to guarantee this is through the use of pull-up resistors on both the SDA and SCL lines. In addition, these pull-ups should be tied to the same voltage source as the VDD pin of the device. In other words is the device VDD is supplied from the main positive supply rail then the SCL and SDA pull-ups should be connected to that same supply rail (as opposed to being connected to a microcontroller I/O pin, for example). Figure 1 is an example of the recommended hardware configuration. The reasoning behind doing this is the same for both adding the pull-up to the SCL line and for utilizing the same supply for the VDD pin and the pull-ups. As anyone who has had any experience with CMOS logic already knows, it is necessary to ensure that all inputs are tied either high or low, since allowing a CMOS input to float can lead to a number of problems. If the SCL line does not have a pull-up, or if the pull-ups are not tied to the VDD supply rail, then conditions occur, however briefly, where the SCL/SDA inputs are floating with respect to the VDD supply voltage. When possible this condition should be avoided.

I²C is a trademark of Philips Semiconductors

When it is not possible to add a pullup resistor to the SCL line (i.e. the hardware design has already been finalized) then the firmware should be configured to either: 1) drive the SCL line high during power up or, 2) float the SCL input during power up.

Of these two options, the first is the recommended method, despite typical concerns regarding latch-up, because it does not negatively impact the battery life in battery powered applications. Microchip Technology's serial EEPROM devices, like all CMOS devices, are susceptible to latch-up, however latch-up does not occur until currents in excess of 100mA are injected into the pin. Typical microcontrollers are not capable of supply currents of this magnitude, therefore the risk of latch-up is extremely low.

The second option is also acceptable but does lead to a brief increase in the current draw of the device during the time period in which the SCL pin is floating with respect to VDD. This increase can be significant in comparison to the normal standby current of the device and can have a detrimental affect on battery life in power sensitive applications.

In all cases it is important that the SCL and SDA lines not be actively held low while the EEPROM device is powered up. This can have an indeterminable effect on the internal state machine and, in some cases, the state machine may fail to correctly initialize and the EEPROM will power up in an incorrect state.

Another improper practice which should be pointed out is the driving of the SDA line high by the microcontroller pin rather than tri-stating the pin and allowing the requisite pullup resistor to pull the bus up to the high state. While this practice would appear harmless enough, and indeed it is as long as the microcontroller and EEPROM device never get out of sync, there is a potential for a high current situation to occur. In the event that the microcontroller and EEPROM should get out of sync, and the EEPROM is outputting a 'low' (i.e. sending an ACK or driving a data bit of '0') while the microcontroller is driving a high then a low impedance path between VDD and VSS is created and excessive current will flow out of the microcontroller I/O pin and into the EEPROM SDA pin. The amount of current that flows is limited only by the IOL specification of the microcontoller's I/O pin. This high current state can obviously have a very detrimental effect on battery life, as well as potentially present long term reliability problems associated with the excess current flow.

FORCING INTERNAL RESET VIA SOFTWARE

In all designs it is recommended that a software reset sequence be sent to the EEPROM as part of the microcontrollers power up sequence. This sequence guarantees that the EEPROM is in a correct and known state. Assuming that the EEPROM has powered up into an incorrect state (or that a reset occurred at the microcontroller during communication), the following sequence (which is further explained below) should be sent in order to guarantee that the serial EEPROM device is properly reset:

- START Bit
- Clock in nine bits of '1'
- START Bit
- STOP Bit

The first START bit will cause the device to reset from a state in which it is expecting to receive data from the microcontroller. In this mode the device is monitoring the data bus in receive mode and can detect the START bit which forces an internal reset.

The nine bits of '1' are used to force a reset of those devices that could not be reset by the previous START bit. This occurs only if the device is in a mode where it is either driving an acknowledge on the bus (low), or is in an output mode and is driving a data bit of '0' out on the bus. In both of these cases the previous START bit (defined as SDA going low while SCL is high) could not be generated due to the device holding the bus low. By sending nine bits of '1' it is guaranteed that the device will see a NACK (microcontroller does not drive the bus low to acknowledge data sent by EEPROM) which also forces an internal reset.

The second START bit is sent to guard against the rare possibility of an erroneous write that could occur if the microcontroller was reset while sending a write command to the EEPROM, and, the EEPROM was driving an ACK on the bus when the first START bit was sent. In this special case if this second START bit was not sent, and instead the STOP bit was sent, the device could initiate a write cycle. This potential for an erroneous write occurs only in the event of the microcontroller being reset while sending a write command to the EEPROM.

The final STOP bit terminates bus activity and puts the EEPROM in standby mode.

This sequence does not effect any other I^2C devices which may be on the bus as they will simply disregard it as an invalid command.

SUMMARY

This application note has presented ideas that are fundamental in nature, yet not always obvious, to the utilization of I^2C serial EEPROM devices. Ideally the hardware/software engineer(s) takes these ideas into consideration during system development and design accordingly. It is recommended that the software reset sequence detailed in this application note be added to the system initilization code of any system that utilizes an I^2C serial EEPROM device.

REFERENCES

'I^2C-Bus Specification', Philips Semiconductors, January 1992

'The I^2C-Bus and How to Use It', Philips Semiconductors, April 1995

TB012

SPI™ 25XX080/160 Mode 1,1 Write Operation

Author: *Shannon Poulin*
Microchip Technology Inc.

INTRODUCTION

The early revision of Microchip's 25xx080 and 25xx160 devices (revision A and B) require a software work around for proper write operation in mode 1,1. Mode 0,0 does NOT require a work around. This issue does not exist in 25xx080 and 25xx160 revision C silicon. The revision number can be found on the outside of a packaged part. It is the middle letter of the three letter string following the package type.

Mode 0,0 and 1,1 indicate that an SPI slave device latches serial data in on the rising edge of clock and out after the falling edge of clock. Mode 0,1 and 1,0 SPI devices would latch data in on the falling edge of clock and out after the rising edge of clock.

Reading a 25xx080 or 25xx160 devices do not require any software adjustments whether it is read in mode 0,0 or 1,1. However, for revision A or B silicon to properly initiate a write, the clock line must be low, when raising chip select ($\overline{CS}$) for the write cycle to begin. In mode 0,0 the polarity of the clock is defined as low, and the clock idles low when not shifting data in or out. In mode 1,1, the polarity of the clock is defined as high, and the clock idles high when not shifting data in or out.

Figure 1 depicts a write timing diagram. It can be seen that for mode 0,0 operation, the clock line is low when $\overline{CS}$ is brought high to start the write cycle timer. The write will be completed successfully in this case.

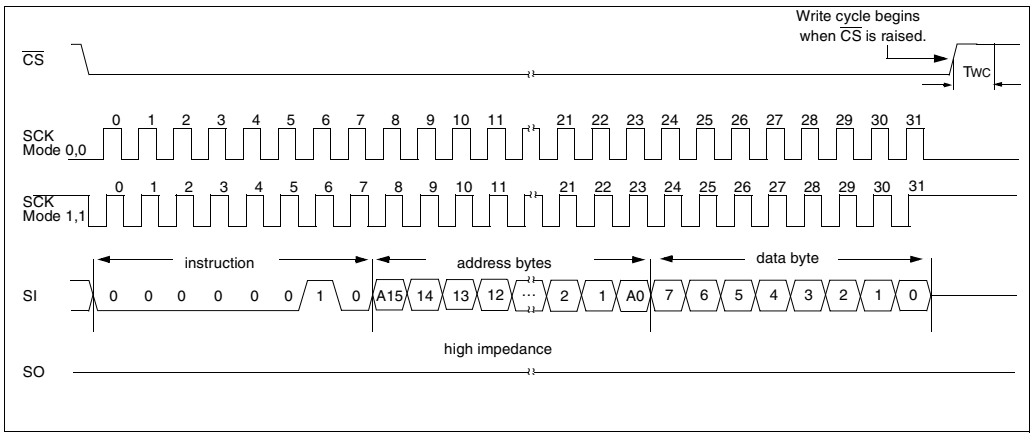
The dashed lines in *Figure 1* also depict the state of the clock in mode 1,1. The drawing shows that the clock line will be high when $\overline{CS}$ is raised. This will not start the write cycle timer, and the part will not write successfully.

Figure 2 shows the timing necessary for proper implementation of mode 1,1 writes. The following steps should be taken to write properly in mode 1,1:

1. Load and send the write instruction.
2. Load and send the address.
3. Load and send the data.
4. **Lower the clock line.**
5. Raise chip select to start a write.
6. **Raise the clock line.**

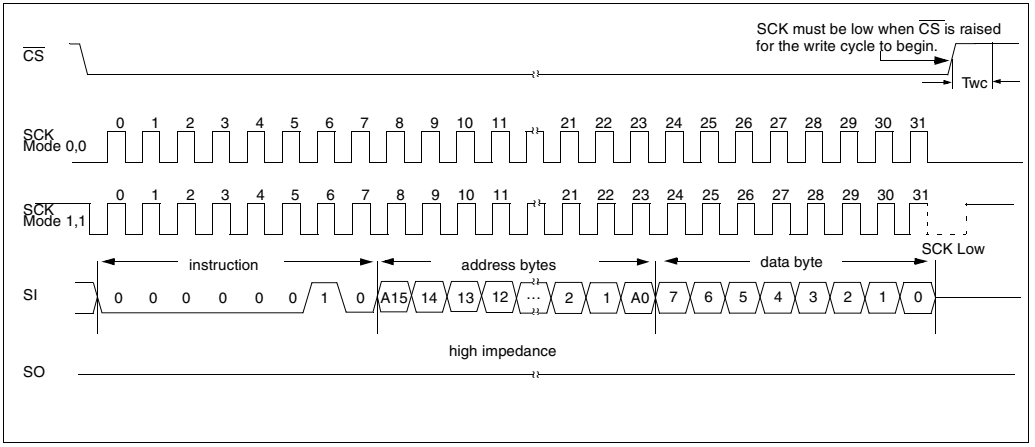
The **bold** indicates the extra steps necessary to properly implement mode 1,1 operation on revision A or B silicon. The write cycle timer will begin when the chip select line is raised.

FIGURE 1: NORMAL WRITE OPERATION



SPI is a trademark of Motorola.

FIGURE 2: MODIFIED WRITE OPERATION



TB014

Operational Differences Between 24LCS21 and 24LCS21A

Author: *Bruce Negley*
Microchip Technology Inc.

DIFFERENCES BETWEEN 24LCS21 AND 24LCS21A

The 24LCS21 and 24LCS21A are both 1K Serial EEPROM devices that implement both DDC1 and DDC2 modes of operation as defined by the VESA committee. Both devices will always power up in DDC1 mode (transmit only), where data is clocked out of the device using the VCLK pin. Both devices will transition to DDC2 mode (bi-directional mode) upon the first high-to-low transition of the SCL line. The only difference between the two devices is that the 24LCS21A has the "return to DDC1" recovery feature as shown in the flowchart (Figure 1). This feature allows the device to return to the DDC1 mode if a valid read or write command is not received in the time that 128 clocks are seen on the VLCK pin.

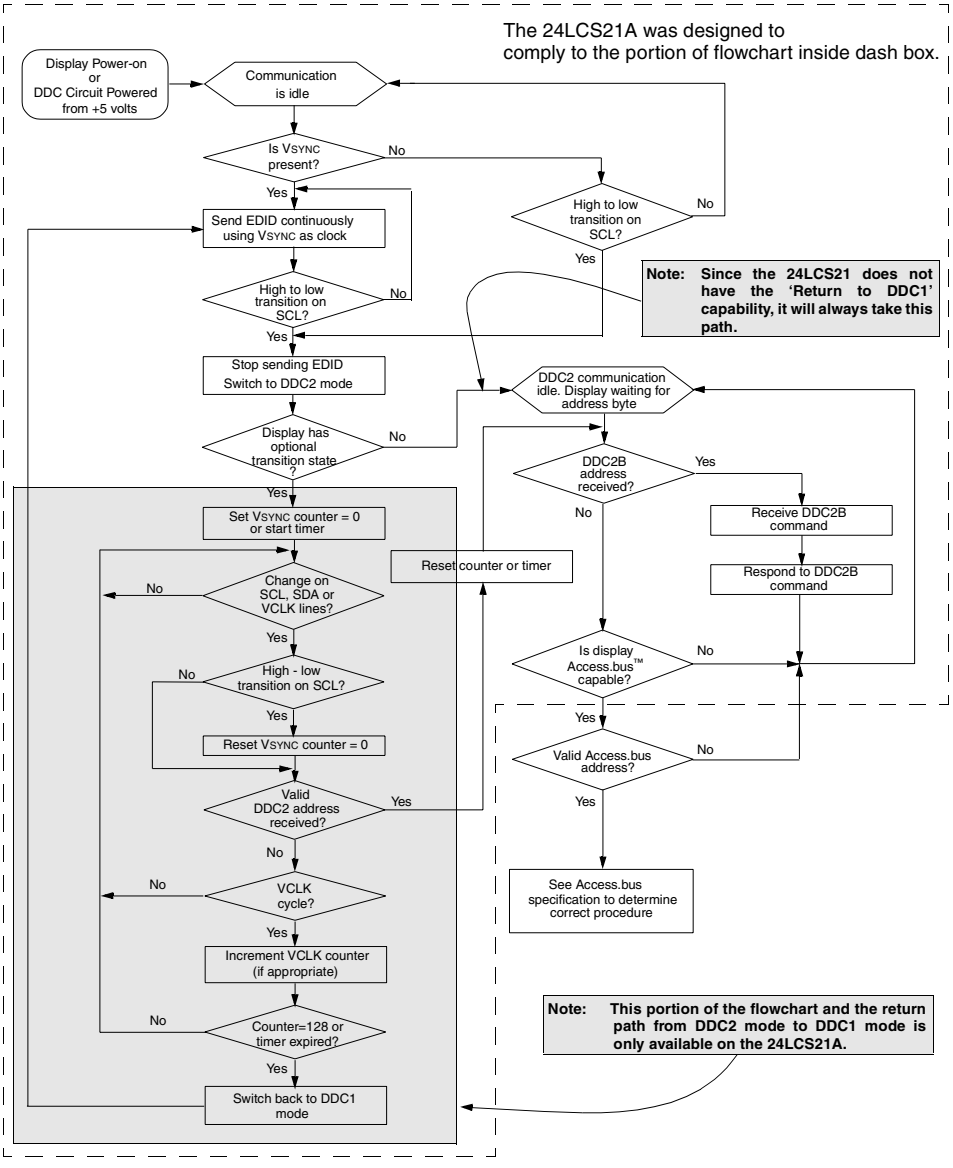
SCENARIO FOR POTENTIAL PROBLEMS BY SWITCHING FROM 24LCS21 TO 24LCS21A

Most video cards will access the EDID table, either with DDC1 mode or DDC2 mode – but not both. If a PC wants to access the EDID table in the monitor using DDC1 mode only, then the two devices will operate exactly the same.

If the PC wants to access the EDID table in DDC2 mode, then the transition to DDC2 mode will be the same for both devices. If this is the operation flow (and it is assumed that most systems will operate this way), both devices will operate in exactly the same way.

The only real issue that can occur by switching to the 24LCS21A device is if the device jumps back to DDC1 while the system still thinks the device is in DDC2 mode. This can only happen if the system toggles the SCL pin to move into DDC2 mode and waits until after the VCLK pin sees 128 clocks before sending the actual command, causing the device to transition back to DDC1. If this happens, then it is possible that the device may not recognize the next transmitted command.

FIGURE 1: FLOWCHART FOR DDC1/DDC2 MODE OPERATION IN 24LCS21A



Note 1: The base flowchart is copyright © 1993, 1994, 1995 Video Electronic Standard Association (VESA) from VESA's Display Data Channel (DDC™) Standard Proposal ver. 2p rev. 0, used by permission of VESA.

2: The dash box and text "The 24LCS21A and …inside dash box." are added by Microchip Technology, Inc.

3: VSYNC signal is normally used to derive a signal for VCLK pin on the 24LCS21A.

SECTION 6
RFID APPLICATION NOTES AND TECHNICAL BRIEFS

AN678

RFID Coil Design

Author: *Youbok Lee*
Microchip Technology Inc.

INTRODUCTION

In a Radio Frequency Identification (RFID) application, an antenna coil is needed for two main reasons:

- To transmit the RF carrier signal to power up the tag
- To receive data signals from the tag

An RF signal can be radiated effectively if the linear dimension of the antenna is comparable with the wavelength of the operating frequency. In an RFID application utilizing the VLF (100 kHz – 500 kHz) band, the wavelength of the operating frequency is a few kilometers (λ = 2.4 Km for 125 kHz signal). Because of its long wavelength, a true antenna can never be formed in a limited space of the device. Alternatively, a small loop antenna coil that is resonating at the frequency of the interest (i.e., 125 kHz) is used. This type of antenna utilizes near field magnetic induction coupling between transmitting and receiving antenna coils.

The field produced by the small dipole loop antenna is not a propagating wave, but rather an attenuating wave. The field strength falls off with r^{-3} (where r = distance from the antenna). This near field behavior (r^{-3}) is a main limiting factor of the read range in RFID applications.

When the time-varying magnetic field is passing through a coil (antenna), it induces a voltage across the coil terminal. This voltage is utilized to activate the passive tag device. The antenna coil must be designed to maximize this induced voltage.

This application note is written as a reference guide for antenna coil designers and application engineers in the RFID industry. It reviews basic electromagnetics theories to understand the antenna coils, a procedure for coil design, calculation and measurement of inductance, an antenna-tuning method, and the relationship between read range vs. size of antenna coil.

REVIEW OF A BASIC THEORY FOR ANTENNA COIL DESIGN

Current and Magnetic Fields

Ampere's law states that current flowing on a conductor produces a magnetic field around the conductor. Figure 1 shows the magnetic field produced by a current element. The magnetic field produced by the current on a round conductor (wire) with a finite length is given by:

EQUATION 1:

$$B_\phi = \frac{\mu_o I}{4\pi r}(\cos\alpha_2 - \cos\alpha_1) \qquad (\text{Weber}/m^2)$$

where:

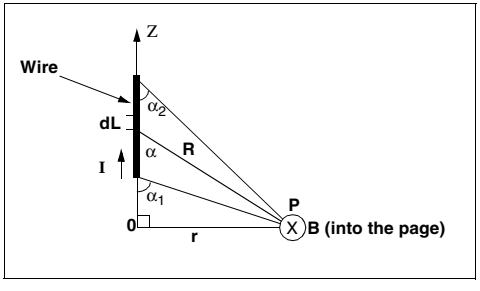
I = current
r = distance from the center of wire
μ_o = permeability of free space and given as $\mu_o = 4\ \pi\ \text{x}\ 10^{-7}$ (Henry/meter)

In a special case with an infinitely long wire where α_1 = -180° and α_2 = 0°, Equation 1 can be rewritten as:

EQUATION 2:

$$B_\phi = \frac{\mu_o I}{2\pi r} \qquad (\text{Weber}/m^2)$$

FIGURE 1: CALCULATION OF MAGNETIC FIELD B AT LOCATION P DUE TO CURRENT I ON A STRAIGHT CONDUCTING WIRE



The magnetic field produced by a circular loop antenna coil with N-turns as shown in Figure 2 is found by:

EQUATION 3:

$$B_z = \frac{\mu_o I N a^2}{2(a^2 + r^2)^{3/2}}$$

$$= \frac{\mu_o I N a^2}{2}\left(\frac{1}{r^3}\right) \quad \text{for } r^2 >> a^2$$

where:

a = radius of loop

Equation 3 indicates that the magnetic field produced by a loop antenna decays with $1/r^3$ as shown in Figure 3. This near-field decaying behavior of the magnetic field is the main limiting factor in the read range of the RFID device. The field strength is maximum in the plane of the loop and directly proportional to the current (I), the number of turns (N), and the surface area of the loop.

Equation 3 is frequently used to calculate the ampere-turn requirement for read range. A few examples that calculate the ampere-turns and the field intensity necessary to power the tag will be given in the following sections.

FIGURE 2: CALCULATION OF MAGNETIC FIELD B AT LOCATION P DUE TO CURRENT I ON THE LOOP

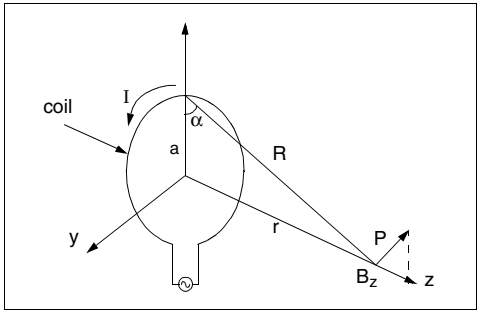
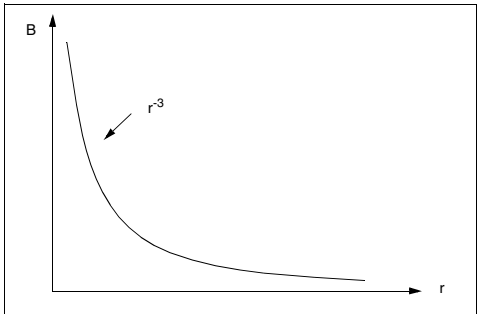


FIGURE 3: DECAYING OF THE MAGNETIC FIELD B VS. DISTANCE *r*



Note: The magnetic field produced by a loop antenna drops off with r^{-3}.

INDUCED VOLTAGE IN ANTENNA COIL

Faraday's law states a time-varying magnetic field through a surface bounded by a closed path induces a voltage around the loop. This fundamental principle has important consequences for operation of passive RFID devices.

Figure 4 shows a simple geometry of an RFID application. When the tag and reader antennas are within a proximity distance, the time-varying magnetic field B that is produced by a reader antenna coil induces a voltage (called electromotive force or simply EMF) in the tag antenna coil. The induced voltage in the coil causes a flow of current in the coil. This is called Faraday's law.

The induced voltage on the tag antenna coil is equal to the time rate of change of the magnetic flux Ψ.

EQUATION 4:

$$V = -N\frac{d\Psi}{dt}$$

where:

- N = number of turns in the antenna coil
- Ψ = magnetic flux through each turn

The negative sign shows that the induced voltage acts in such a way as to oppose the magnetic flux producing it. This is known as Lenz's Law and it emphasizes the fact that the direction of current flow in the circuit is such that the induced magnetic field produced by the induced current will oppose the original magnetic field.

The magnetic flux Ψ in Equation 4 is the total magnetic field B that is passing through the entire surface of the antenna coil, and found by:

EQUATION 5:

$$\psi = \int B \cdot dS$$

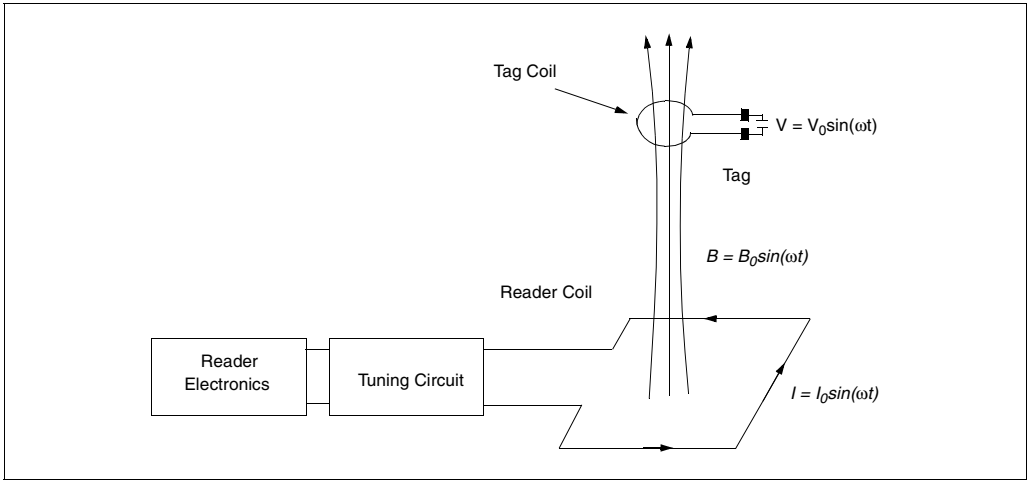
where:

- B = magnetic field given in Equation 3
- S = surface area of the coil
- • = inner product (*cosine angle between two vectors*) of vectors B and surface area S

Note:	Both magnetic field $\boldsymbol{B}$ and surface $\boldsymbol{S}$ are vector quantities.

The inner product presentation of two vectors in Equation 5 suggests that the total magnetic flux ψ that is passing through the antenna coil is affected by an orientation of the antenna coils. The inner product of two vectors becomes maximized when the two vectors are in the same direction. Therefore, the magnetic flux that is passing through the tag coil will become maximized when the two coils (reader coil and tag coil) are placed in parallel with respect to each other.

FIGURE 4: A BASIC CONFIGURATION OF READER AND TAG ANTENNAS IN AN RFID APPLICATION



From Equations 3, 4, and 5, the induced voltage V_0 for an untuned loop antenna is given by:

EQUATION 6:

$$V_o = 2\pi f N S B_o \cos\alpha$$

where:

f	=	frequency of the arrival signal
N	=	number of turns of coil in the loop
S	=	area of the loop in square meters (m^2)
B_o	=	strength of the arrival signal
α	=	angle of arrival of the signal

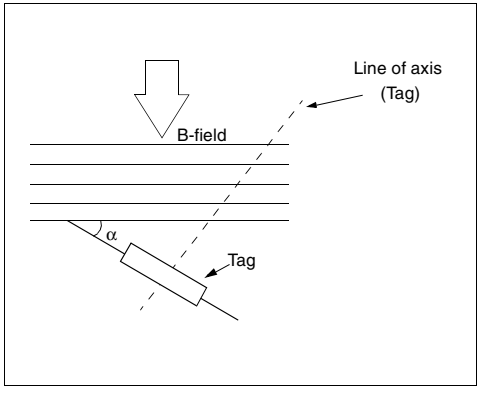
If the coil is tuned (with capacitor C) to the frequency of the arrival signal (125 kHz), the output voltage V_o will rise substantially. The output voltage found in Equation 6 is multiplied by the loaded Q (Quality Factor) of the tuned circuit, which can be varied from 5 to 50 in typical low-frequency RFID applications:

EQUATION 7:

$$V_o = 2\pi f_o N Q S B_o \cos\alpha$$

where the loaded Q is a measure of the selectivity of the frequency of the interest. The Q will be defined in Equations 30, 31, and 37 for general, parallel, and serial resonant circuit, respectively.

FIGURE 5: ORIENTATION DEPENDENCY OF THE TAG ANTENNA.



The induced voltage developed across the loop antenna coil is a function of the angle of the arrival signal. The induced voltage is maximized when the antenna coil is placed perpendicular to the direction of the incoming signal where $\alpha = 0$.

EXAMPLE 1: B-FIELD REQUIREMENT

The strength of the B-field that is needed to turn on the tag can be calculated from Equation 7:

EQUATION 8:

$$B_o = \frac{Vo}{2\pi f_o N Q S \cos\alpha}$$

$$= \frac{7(2.4)}{(2\pi)(125\text{ kHz})(100)(15)(38.71\text{cm}^2)}$$

$$\approx 1.5 \quad \mu\text{Wb/m}^2$$

where the following parameters are used in the above calculation:

tag coil size	=	2 x 3 inches = 38.71 cm^2: (credit card size)
frequency	=	125 kHz
number of turns	=	100
Q of antenna coil	=	15
AC coil voltage to turn on the tag	=	7 V
cos α	=	1 (normal direction, $\alpha = 0$).

EXAMPLE 2: NUMBER OF TURNS AND CURRENT (AMPERE-TURNS) OF READER COIL

Assuming that the reader should provide a read range of 10 inches (25.4 cm) with a tag given in Example 1, the requirement for the current and number of turns (Ampere-turns) of a reader coil that has an 8 cm radius can be calculated from Equation 3:

EQUATION 9:

$$(NI) = \frac{2B_z(a^2 + r^2)^{3/2}}{\mu a^2}$$

$$= \frac{2(1.5\times10^{-6})(0.08^2 + 0.254^2)^{3/2}}{(4\pi\times10^{-7})(0.08)}$$

$$= 7.04 \text{ (ampere - turns)}$$

This is an attainable number. If, however, we wish to have a read range of 20 inches (50.8 cm), it can be found that NI increases to 48.5 ampere-turns. At 25.2 inches (64 cm), it exceeds 100 ampere-turns.

For a longer read range, it is instructive to consider increasing the radius of the coil. For example, by doubling the radius (16 cm) of the loop, the ampere-turns requirement for the same read range (10 inches: 25.4 cm) becomes:

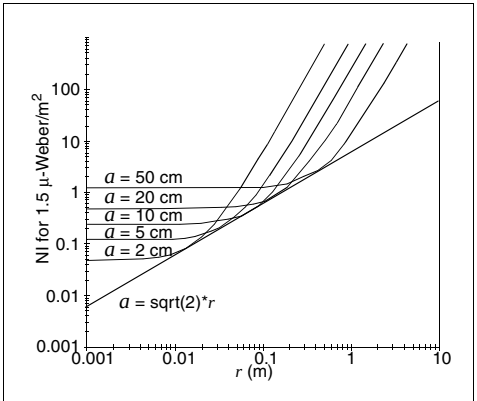
EQUATION 10:

$$NI = \frac{2(1.5\times10^{-6})(0.16^2+0.25^2)^{3/2}}{(4\pi\times10^{-7})(0.16^2)}$$

$$= 2.44 \text{ (ampere-turns)}$$

At a read range of 20 inches (50.8 cm), the ampere-turns becomes 13.5 and at 25.2 inches (64 cm), 26.8. Therefore, for a longer read range, increasing the tag size is often more effective than increasing the coil current. Figure 6 shows the relationship between the read range and the ampere-turns (*IN*).

FIGURE 6: AMPERE-TURNS VS. READ RANGE FOR AN ACCESS CONTROL CARD (CREDIT CARD SIZE)



Note: B_O = 1.5 μWb/m^2 is used.

The optimum radius of loop that requires the minimum number of ampere-turns for a particular read range can be found from Equation 3 such as:

EQUATION 11:

$$NI = K\frac{(a^2+r^2)^{\frac{3}{2}}}{a^2}$$

where:

$$K = \frac{2B_z}{\mu_o}$$

By taking derivative with respect to the radius *a*,

$$\frac{d(NI)}{da} = K\frac{3/2(a^2+r^2)^{1/2}(2a^3)-2a(a^2+r^2)^{3/2}}{a^4}$$

$$= K\frac{(a^2-2r^2)(a^2+r^2)^{1/2}}{a^3}$$

The above equation becomes minimized when:

$$a^2-2r^2 = 0$$

The above result shows a relationship between the read range vs. tag size. The optimum radius is found as:

$$a = \sqrt{2}r$$

where:

a = radius of coil
r = read range

The above result indicates that the optimum radius of loop for a reader antenna is 1.414 times the read range *r*.

WIRE TYPES AND OHMIC LOSSES

Wire Size and DC Resistance

The diameter of electrical wire is expressed as the American Wire Gauge (AWG) number. The gauge number is inversely proportional to diameter and the diameter is roughly doubled every six wire gauges. The wire with a smaller diameter has higher DC resistance. The DC resistance for a conductor with a uniform cross-sectional area is found by:

EQUATION 12:

$$R_{DC} = \frac{l}{\sigma S} \qquad (\Omega)$$

where:

l = total length of the wire
σ = conductivity
S = cross-sectional area

Table 1 shows the diameter for bare and enamel-coated wires, and DC resistance.

AC Resistance of Wire

At DC, charge carriers are evenly distributed through the entire cross section of a wire. As the frequency increases, the reactance near the center of the wire increases. This results in higher impedance to the current density in the region. Therefore, the charge moves away from the center of the wire and towards the edge of the wire. As a result, the current density decreases in the center of the wire and increases near the edge of the wire. This is called a *skin effect*. The depth into the conductor at which the current density falls to 1/e, or 37% of its value along the surface, is known as the *skin depth* and is a function of the frequency and the permeability and conductivity of the medium. The skin depth is given by:

EQUATION 13:

$$\delta = \frac{1}{\sqrt{\pi f \mu \sigma}}$$

where:

f = frequency
μ = permeability of material
σ = conductivity of the material

EXAMPLE 3:

The skin depth for a copper wire at 125 kHz can be calculated as:

EQUATION 14:

$$\delta = \frac{1}{\sqrt{\pi f (4\pi \times 10^{-7})(5.8 \times 10^{-7})}}$$

$$= \frac{0.06608}{\sqrt{f}} \qquad (m)$$

$$= 0.187 \qquad (mm)$$

The wire resistance increases with frequency, and the resistance due to the skin depth is called an AC resistance. An approximated formula for the ac resistance is given by:

EQUATION 15:

$$R_{ac} \approx \frac{1}{2\sigma\pi\delta} = (R_{DC})\frac{a}{2\delta} \qquad (\Omega)$$

where:

a = coil radius

For copper wire, the loss is approximated by the DC resistance of the coil, if the wire radius is greater than $.066/\sqrt{f}$ cm. At 125 kHz, the critical radius is 0.019 cm. This is equivalent to #26 gauge wire. Therefore, for minimal loss, wire gauge numbers of greater than #26 should be avoided if coil Q is to be maximized.

TABLE 1: AWG WIRE CHART

Wire Size (AWG)	Dia. in Mils (bare)	Dia. in Mils (coated)	Ohms/ 1000 ft.	Cross Section (mils)
1	289.3	—	0.126	83690
2	287.6	—	0.156	66360
3	229.4	—	0.197	52620
4	204.3	—	0.249	41740
5	181.9	—	0.313	33090
6	162.0	—	0.395	26240
7	166.3	—	0.498	20820
8	128.5	131.6	0.628	16510
9	114.4	116.3	0.793	13090
10	101.9	106.2	0.999	10380
11	90.7	93.5	1.26	8230
12	80.8	83.3	1.59	6530
13	72.0	74.1	2.00	5180
14	64.1	66.7	2.52	4110
15	57.1	59.5	3.18	3260
16	50.8	52.9	4.02	2580
17	45.3	47.2	5.05	2060
18	40.3	42.4	6.39	1620
19	35.9	37.9	8.05	1290
20	32.0	34.0	10.1	1020
21	28.5	30.2	12.8	812
22	25.3	28.0	16.2	640
23	22.6	24.2	20.3	511
24	20.1	21.6	25.7	404
25	17.9	19.3	32.4	320

Note: 1 mil = 2.54 x 10^{-3} cm

Wire Size (AWG)	Dia. in Mils (bare)	Dia. in Mils (coated)	Ohms/ 1000 ft.	Cross Section (mils)
26	15.9	17.2	41.0	253
27	14.2	15.4	51.4	202
28	12.6	13.8	65.3	159
29	11.3	12.3	81.2	123
30	10.0	11.0	106.0	100
31	8.9	9.9	131	79.2
32	8.0	8.8	162	64.0
33	7.1	7.9	206	50.4
34	6.3	7.0	261	39.7
35	5.6	6.3	331	31.4
36	5.0	5.7	415	25.0
37	4.5	5.1	512	20.2
38	4.0	4.5	648	16.0
39	3.5	4.0	847	12.2
40	3.1	3.5	1080	9.61
41	2.8	3.1	1320	7.84
42	2.5	2.8	1660	6.25
43	2.2	2.5	2140	4.84
44	2.0	2.3	2590	4.00
45	1.76	1.9	3350	3.10
46	1.57	1.7	4210	2.46
47	1.40	1.6	5290	1.96
48	1.24	1.4	6750	1.54
49	1.11	1.3	8420	1.23
50	0.99	1.1	10600	0.98

Note: 1 mil = 2.54 x 10^{-3} cm

INDUCTANCE OF VARIOUS ANTENNA COILS

The electrical current flowing through a conductor produces a magnetic field. This time-varying magnetic field is capable of producing a flow of current through another conductor. This is called inductance. The inductance L depends on the physical characteristics of the conductor. A coil has more inductance than a straight wire of the same material, and a coil with more turns has more inductance than a coil with fewer turns. The inductance L of inductor is defined as the ratio of the total magnetic flux linkage to the current I through the inductor: i.e.,

EQUATION 16:

$$L = \frac{N\psi}{I} \qquad \text{(Henry)}$$

where:

N = number of turns
I = current
Ψ = magnetic flux

In a typical RFID antenna coil for 125 kHz, the inductance is often chosen as a few (mH) for a tag and from a few hundred to a few thousand (µH) for a reader. For a coil antenna with multiple turns, greater inductance results with closer turns. Therefore, the tag antenna coil that has to be formed in a limited space often needs a multi-layer winding to reduce the number of turns.

The design of the inductor would seem to be a relatively simple matter. However, it is almost impossible to construct an ideal inductor because:

a) The coil has a finite conductivity that results in losses, and
b) The distributed capacitance exists between turns of a coil and between the conductor and surrounding objects.

The actual inductance is always a combination of resistance, inductance, and capacitance. The apparent inductance is the effective inductance at any frequency, i.e., inductive minus the capacitive effect. Various formulas are available in literatures for the calculation of inductance for wires and coils[1, 2].

The parameters in the inductor can be measured. For example, an HP 4285 Precision LCR Meter can measure the inductance, resistance, and Q of the coil.

Inductance of a Straight Wire

The inductance of a straight wound wire shown in Figure 1 is given by:

EQUATION 17:

$$L = 0.002l\left[\log_e\frac{2l}{a} - \frac{3}{4}\right] \qquad (\mu H)$$

where:

l and a = length and radius of wire in cm, respectively.

EXAMPLE 4: CALCULATION OF INDUCTANCE FOR A STRAIGHT WIRE

The inductance of a wire with 10 feet (304.8 cm) long and 2 mm diameter is calculated as follows:

EQUATION 18:

$$L = 0.002(304.8)\left[\ln\left(\frac{2(304.8)}{0.1}\right) - \frac{3}{4}\right]$$

$$= 0.60967(7.965)$$

$$= 4.855(\mu H)$$

Inductance of a Single Layer Coil

The inductance of a single layer coil shown in Figure 7 can be calculated by:

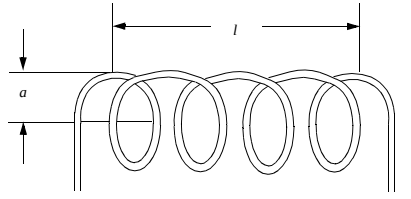
EQUATION 19:

$$L = \frac{(aN)^2}{22.9l + 25.4a} \qquad (\mu H)$$

where:

a = coil radius (cm)
l = coil length (cm)
N = number of turns

FIGURE 7: A SINGLE LAYER COIL



Note: For best Q of the coil, the length should be roughly the same as the diameter of the coil.

Inductance of a Circular Loop Antenna Coil with Multilayer

To form a big inductance coil in a limited space, it is more efficient to use multilayer coils. For this reason, a typical RFID antenna coil is formed in a planar multi-turn structure. Figure 8 shows a cross section of the coil. The inductance of a circular ring antenna coil is calculated by an empirical formula[2]:

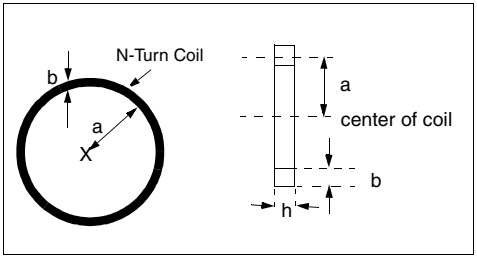
EQUATION 20:

$$L = \frac{0.31(aN)^2}{6a + 9h + 10b} \qquad (\mu H)$$

where:

- a = average radius of the coil in cm
- N = number of turns
- b = winding thickness in cm
- h = winding height in cm

FIGURE 8: A CIRCULAR LOOP AIR CORE ANTENNA COIL WITH N-TURNS



The number of turns needed for a certain inductance value is simply obtained from Equation 20 such that:

EQUATION 21:

$$N = \sqrt{\frac{L_{\mu H}(6a + 9h + 10b)}{(0.31)a^2}}$$

EXAMPLE 5: EXAMPLE ON NUMBER OF TURNS

Equation 21 results in N = 200 turns for L = 3.87 mH with the following coil geometry:

- a = 1 inch (2.54 cm)
- h = 0.05 cm
- b = 0.5 cm

To form a resonant circuit for 125 kHz, it needs a capacitor across the inductor. The resonant capacitor can be calculated as:

EQUATION 22:

$$C = \frac{1}{(2\pi f)^2 L} = \frac{1}{(4\pi^2)(125\times10^3)(3.87\times10^{-3})}$$

$$= 419 \qquad (pF)$$

Inductance of a Square Loop Coil with Multilayer

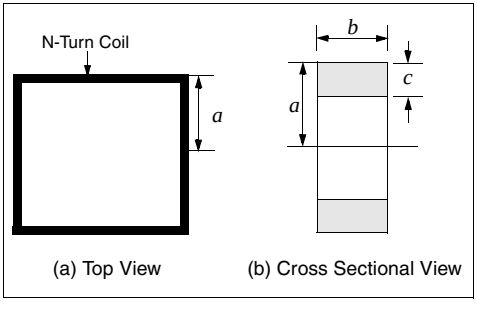
If N is the number of turns and a is the side of the square measured to the center of the rectangular cross section that has length b and depth c as shown in Figure 9, then[2]:

EQUATION 23:

$$L = 0.008aN^2\left(2.303\log_{10}\left(\frac{a}{b+c}\right) + 0.2235\frac{b+c}{a} + 0.726\right) \quad (\mu H)$$

The formulas for inductance are widely published and provide a reasonable approximation for the relationship between inductance and number of turns for a given physical size[1]-[4]. When building prototype coils, it is wise to exceed the number of calculated turns by about 10%, and then remove turns to achieve resonance. For production coils, it is best to specify an inductance and tolerance rather than a specific number of turns.

FIGURE 9: A SQUARE LOOP ANTENNA COIL WITH MULTILAYER



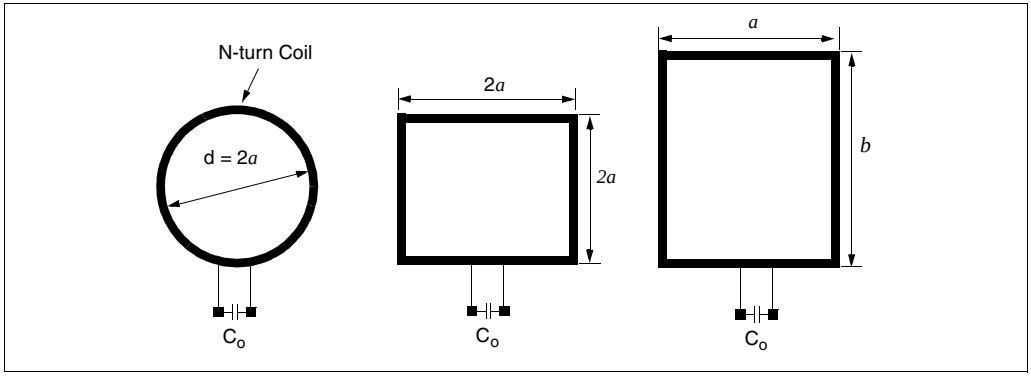
CONFIGURATION OF ANTENNA COILS

Tag Antenna Coil

An antenna coil for an RFID tag can be configured in many different ways, depending on the purpose of the application and the dimensional constraints. A typical inductance L for the tag coil is a few (mH) for 125 kHz devices. Figure 10 shows various configurations of tag antenna coils. The coil is typically made of a thin wire. The inductance and the number of turns of the coil can be calculated by the formulas given in the previous section. An Inductance Meter is often used to measure the inductance of the coil. A typical number of turns of the coil is in the range of 100 turns for 125 kHz and 3~5 turns for 13.56 MHz devices.

For a longer read range, the antenna coil must be tuned properly to the frequency of interest (i.e., 125 kHz). Voltage drop across the coil is maximized by forming a parallel resonant circuit. The tuning is accomplished with a resonant capacitor that is connected in parallel to the coil as shown in Figure 10. The formula for the resonant capacitor value is given in Equation 22.

FIGURE 10: VARIOUS CONFIGURATIONS OF TAG ANTENNA COIL



Reader Antenna Coil

The inductance for the reader antenna coil is typically in the range of a few hundred to a few thousand micro-Henries (μH) for low frequency applications. The reader antenna can be made of either a single coil that is typically forming a series resonant circuit or a double loop (transformer) antenna coil that forms a parallel resonant circuit.

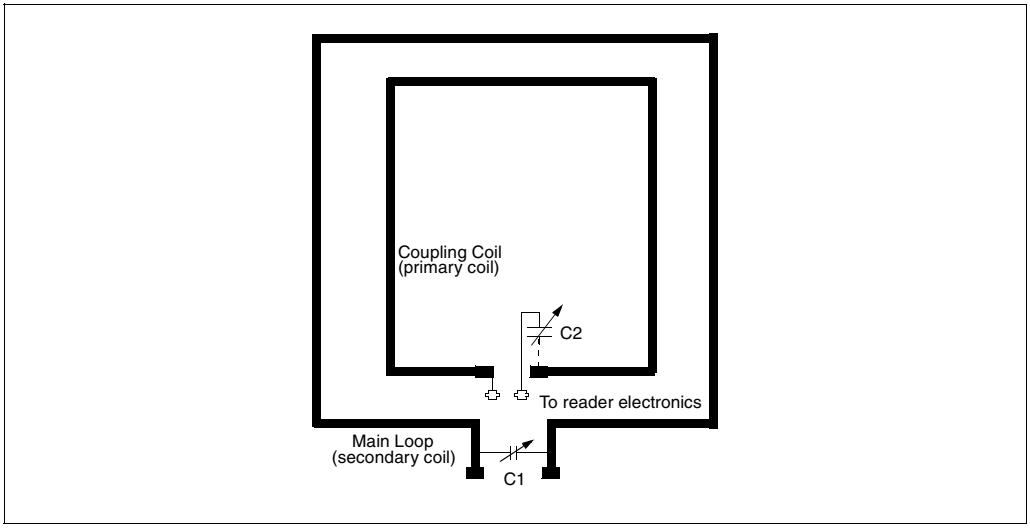
The series resonant circuit results in minimum impedance at the resonance frequency. Therefore, it draws a maximum current at the resonance frequency. On the other hand, the parallel resonant circuit results in maximum impedance at the resonance frequency. Therefore, the current becomes minimized at the resonance frequency. Since the voltage can be stepped up by forming a double loop (parallel) coil, the parallel resonant circuit is often used for a system where a higher voltage signal is required.

Figure 11 shows an example of the transformer loop antenna. The main loop (secondary) is formed with several turns of wire on a large frame, with a tuning capacitor to resonate it to the resonance frequency (125 kHz). The other loop is called a coupling loop (primary), and it is formed with less than two or three turns of coil. This loop is placed in a very close proximity to the main loop, usually (but not necessarily) on the inside edge and not more than a couple of centimeters away from the main loop. The purpose of this loop is to couple signals induced from the main loop to the reader (or vise versa) at a more reasonable matching impedance.

The coupling (primary) loop provides an impedance match to the input/output impedance of the reader. The coil is connected to the input/output signal driver in the reader electronics. The main loop (secondary) must be tuned to resonate at the resonance frequency and is not physically connected to the reader electronics.

The coupling loop is usually untuned, but in some designs, a tuning capacitor C2 is placed in series with the coupling loop. Because there are far fewer turns on the coupling loop than the main loop, its inductance is considerably smaller. As a result, the capacitance to resonate is usually much larger.

FIGURE 11: A TRANSFORMER LOOP ANTENNA FOR READER



RESONANCE CIRCUITS, QUALITY FACTOR *Q*, AND BANDWIDTH

In RFID applications, the antenna coil is an element of resonant circuit and the read range of the device is greatly affected by the performance of the resonant circuit.

Figures 12 and 13 show typical examples of resonant circuits formed by an antenna coil and a tuning capacitor. The resonance frequency (f_o) of the circuit is determined by:

EQUATION 24:

$$f_o = \frac{1}{2\pi\sqrt{LC}}$$

where:

L = inductance of antenna coil

C = tuning capacitance

The resonant circuit can be formed either series or parallel.

The series resonant circuit has a minimum impedance at the resonance frequency. As a result, maximum current is available in the circuit. This series resonant circuit is typically used for the reader antenna.

On the other hand, the parallel resonant circuit has maximum impedance at the resonance frequency. It offers minimum current and maximum voltage at the resonance frequency. This parallel resonant circuit is used for the tag antenna.

Parallel Resonant Circuit

Figure 12 shows a simple parallel resonant circuit. The total impedance of the circuit is given by:

EQUATION 25:

$$Z(j\omega) = \frac{j\omega L}{(1-\omega^2 LC) + j\frac{\omega L}{R}} \qquad (\Omega)$$

where:

ω = angular frequency = 2πf

R = load resistor

The ohmic resistance r of the coil is ignored. The maximum impedance occurs when the denominator in the above equation minimized such as:

EQUATION 26:

$$\omega^2 LC = 1$$

This is called a resonance condition and the resonance frequency is given by:

EQUATION 27:

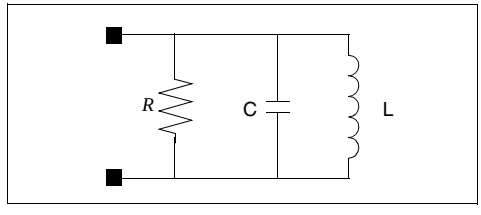
$$f_o = \frac{1}{2\pi\sqrt{LC}}$$

By applying Equation 26 into Equation 25, the impedance at the resonance frequency becomes:

EQUATION 28:

$$Z = R$$

FIGURE 12: PARALLEL RESONANT CIRCUIT



The R and C in the parallel resonant circuit determine the bandwidth, B, of the circuit.

EQUATION 29:

$$B = \frac{1}{2\pi RC} \qquad (Hz)$$

The quality factor, Q, is defined by various ways such as:

EQUATION 30:

$$Q = \frac{\text{Energy Stored in the System per One Cycle}}{\text{Energy Dissipated in the System per One Cycle}}$$

$$= \frac{f_o}{B}$$

where:

f_o = resonant frequency
B = bandwidth

By applying Equation 27 and Equation 29 into Equation 30, the loaded Q in the parallel resonant circuit is:

EQUATION 31:

$$Q = R\sqrt{\frac{C}{L}}$$

The Q in parallel resonant circuit is directly proportional to the load resistor R and also to the square root of the ratio of capacitance and inductance in the circuit.

When this parallel resonant circuit is used for the tag antenna circuit, the voltage drop across the circuit can be obtained by combining Equations 7 and 31,

EQUATION 32:

$$V_o = 2\pi f_o NQSB_o \cos\alpha$$

$$= 2\pi f_o N\left(R\sqrt{\frac{C}{L}}\right)SB_o \cos\alpha$$

The above equation indicates that the induced voltage in the tag coil is inversely proportional to the square root of the coil inductance, but proportional to the number of turns and surface area of the coil.

The parallel resonant circuit can be used in the transformer loop antenna for a long-range reader as discussed in "Reader Antenna Coil" (Figure 11). The voltage in the secondary loop is proportional to the turn ratio (n_2/n_1) of the transformer loop. However, this high voltage signal can corrupt the receiving signals. For this reason, a separate antenna is needed for receiving the signal. This receiving antenna circuit should be tuned to the modulating signal of the tag and detunned to the carrier signal frequency for maximum read range.

Series Resonant Circuit

A simple series resonant circuit is shown in Figure 13. The expression for the impedance of the circuit is:

EQUATION 33:

$$Z(j\omega) = r + j(X_L - X_C) \qquad (\Omega)$$

where:

r = ohmic resistance of the circuit

EQUATION 34:

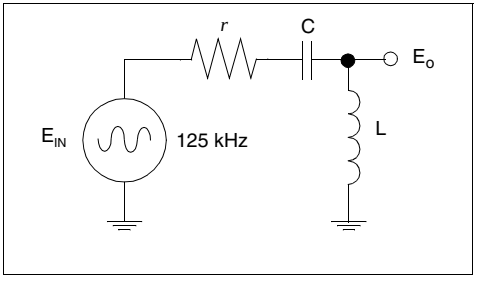
$$X_L = 2\pi f_o L \qquad (\Omega)$$

EQUATION 35:

$$X_c = \frac{1}{2\pi f_o C} \qquad (\Omega)$$

The impedance in Equation 33 becomes minimized when the reactance component cancelled out each other such that $X_L = X_C$. This is called a resonance condition. The resonance frequency is same as the parallel resonant frequency given in Equation 27.

FIGURE 13: SERIES RESONANCE CIRCUIT



The half power frequency bandwidth is determined by r and L, and given by:

EQUATION 36:

$$\boldsymbol{B} = \frac{r}{2\pi L} \qquad (Hz)$$

The quality factor, Q, in the series resonant circuit is given by:

EQUATION 37:

$$Q = \frac{f_o}{B} = \begin{cases} \frac{\omega L}{r} = \frac{1}{\omega C r} & \text{; for unloaded circuit} \\ \frac{1}{r}\sqrt{\frac{L}{C}} & \text{; for loaded circuit} \end{cases}$$

The series circuit forms a voltage divider; the voltage drops in the coil is given by:

EQUATION 38:

$$V_o = \frac{jX_L}{r + jX_L - jX_C} V_{in}$$

or

EQUATION 39:

$$\left|\frac{V_o}{V_{in}}\right| = \frac{X_L}{\sqrt{r^2 + (X_L - X_c)^2}} = \frac{X_L}{r\sqrt{1 + \left(\frac{X_L - X_c}{r}\right)^2}} = \frac{Q}{\sqrt{1 + \left(\frac{X_L - X_c}{r}\right)^2}}$$

EXAMPLE 6: CIRCUIT PARAMETERS.

If the series resistance of the circuit is 15 Ω, then the L and C values form a 125 kHz resonant circuit with Q = 8 are:

EQUATION 40:

$$X_L = Qr_s = 120\Omega$$

$$L = \frac{X_L}{2\pi f} = \frac{120}{2\pi(125\text{ kHz})} = 153 \quad (\mu H)$$

$$C = \frac{1}{2\pi f X_L} = \frac{1}{2\pi(125\text{ kHz})(120)} = 10.6 \quad (nF)$$

EXAMPLE 7: CALCULATION OF READ RANGE

Let us consider designing a reader antenna coil with L = 153 µH, diameter = 10 cm, and winding thickness and height are small compared to the diameter.

The number of turns for the inductance can be calculated from Equation 21, resulting in 24 turns.

If the current flow through the coil is 0.5 amperes, the ampere-turns becomes 12. Therefore, the read range for this coil will be about 20 cm with a credit card size tag.

Q and Bandwidth

Figure 14 shows the approximate frequency bands for common forms of Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK), and Phase Shift Keying (PSK) modulation. For a full recovery of data signal from the tag, the reader circuit needs a bandwidth that is at least twice the data rate. Therefore, if the data rate is 8 kHz for an ASK signal, the bandwidth must be at least 16 kHz for a full recovery of the information that is coming from the tag.

The data rate for FSK (÷ 10) signal is 12.5 kHz. Therefore, a bandwidth of 25 kHz is needed for a full data recovery.

The *Q* for this FSK (÷ 10) signal can be obtained from Equation 30.

EQUATION 41:

$$Q = \frac{f_o}{B} = \frac{125\ \text{kHz}}{25\ \text{kHz}}$$

$$= 5$$

For a PSK (÷ 2) signal, the data rate is 62.5 kHz (if the carrier frequency is 125 kHz) therefore, the reader circuit needs 125 kHz of bandwidth. The *Q* in this case is 1, and consequently the circuit becomes *Q*-independent.

This problem may be solved by separating the transmitting and receiving coils. The transmitting coil can be designed with higher *Q* and the receiving coil with lower *Q*.

Limitation on *Q*

When designing a reader antenna circuit, the temptation is to design a coil with very high *Q*. There are three important limitations to this approach.

a) Very high voltages can cause insulation breakdown in either the coil or resonant capacitor.

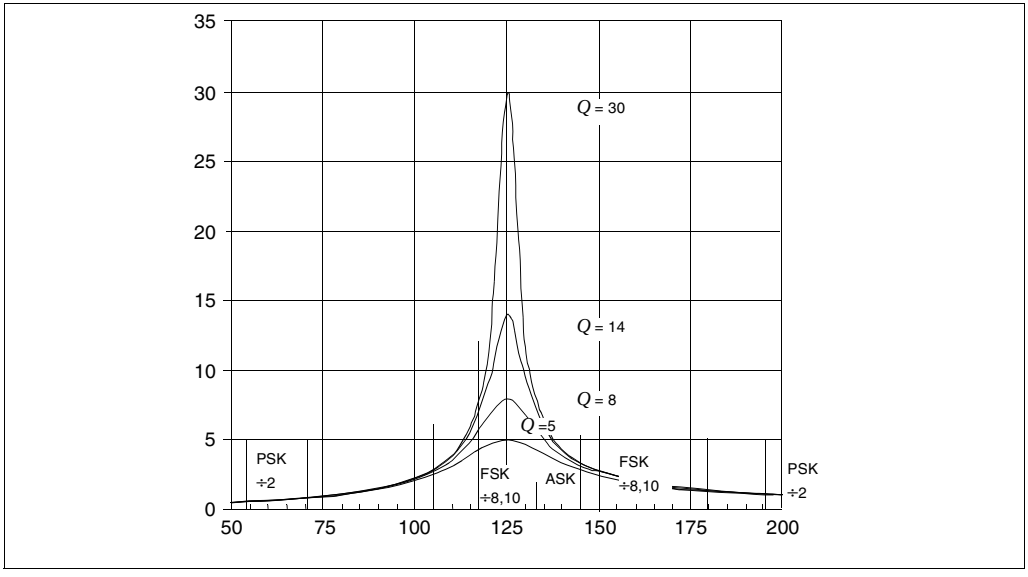
   For example, a 1 ampere of current flow in a 2 mH coil will produce a voltage drop of 1500 VPP. Such voltages are easy to obtain but difficult to isolate. In addition, in the case of single coil reader designs, recovery of the return signal from the tag must be accomplished in the presence of these high voltages.

b) Tuning becomes critical.

   To implement a high *Q* antenna circuit, high voltage components with a close tolerance and high stability would have to be used. Such parts are generally expensive and difficult to obtain.

c) As the *Q* of the circuit gets higher, the amplitude of the return signal relative to the power of the carrier gets proportionally smaller complicating its recovery by the reader circuit.

FIGURE 14: *Q* FACTOR VS. MODULATION SIGNALS



Tuning Method

The circuit must be tuned to the resonance frequency for a maximum performance (read range) of the device. Two examples of tuning the circuit are as follows:

- **Voltage Measurement Method:**
  a) Set up a voltage signal source at the resonance frequency (125 kHz)
  b) Connect a voltage signal source across the resonant circuit.
  c) Connect an Oscilloscope across the resonant circuit.
  d) Tune the capacitor or the coil while observing the signal amplitude on the Oscilloscope.
  e) Stop the tuning at the maximum voltage.

- **S-parameter or Impedance Measurement Method using Network Analyzer:**
  a) Set up an S-Parameter Test Set (Network Analyzer) for S11 measurement, and do a calibration.
  b) Measure the S11 for the resonant circuit.
  c) Reflection impedance or reflection admittance can be measured instead of the S11.
  d) Tune the capacitor or the coil until a maximum null (S11) occurs at the resonance frequency, f_o. For the impedance measurement, the maximum peak will occur for the parallel resonant circuit, and minimum peak for the series resonant circuit.

FIGURE 15: VOLTAGE VS. FREQUENCY FOR RESONANT CIRCUIT

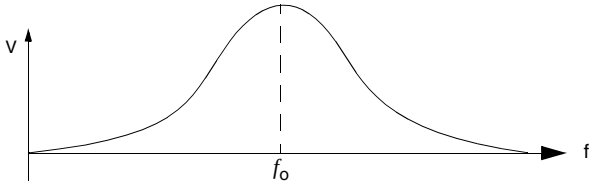
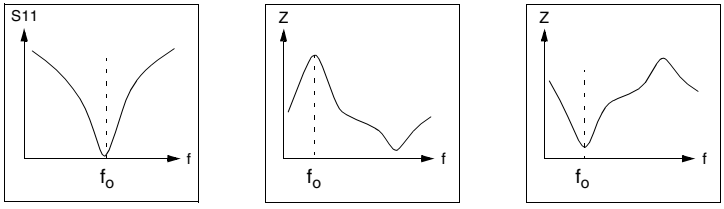


FIGURE 16: FREQUENCY RESPONSES FOR RESONANT CIRCUIT



(a) (b) (c)

Note 1: (a) S11 Response, (b) Impedance Response for a Parallel Resonant Circuit, and (c) Impedance Response for a Series Resonant Circuit.

2: In (a), the null at the resonance frequency represents a minimum input reflection at the resonance frequency. This means the circuit absorbs the signal at the frequency while other frequencies are reflected back. In (b), the impedance curve has a peak at the resonance frequency. This is because the parallel resonant circuit has a maximum impedance at the resonance frequency. (c) shows a response for the series resonant circuit. Since the series resonant circuit has a minimum impedance at the resonance frequency, a minimum peak occurs at the resonance frequency.

READ RANGE OF RFID DEVICES

Read range is defined as a maximum communication distance between the reader and tag. The read range of typical passive RFID products varies from about 1 inch to 1 meter, depending on system configuration. The read range of an RFID device is, in general, affected by the following parameters:

a) Operating frequency and performance of antenna coils
b) Q of antenna and tuning circuit
c) Antenna orientation
d) Excitation current and voltage
e) Sensitivity of receiver
f) Coding (or modulation) and decoding (or demodulation) algorithm
g) Number of data bits and detection (interpretation) algorithm
h) Condition of operating environment (metallic, electrical noise), etc.

With a given operating frequency, the above conditions (a – c) are related to the antenna configuration and tuning circuit. The conditions (d – e) are determined by a circuit topology of the reader. The condition (f) is called the communication protocol of the device, and (g) is related to a firmware program for data interpretation.

Assuming the device is operating under a given condition, the read range of the device is largely affected by the performance of the antenna coil. It is always true that a longer read range is expected with the larger size of the antenna. Figures 17 and 18 show typical examples of the read range of various passive RFID devices.

FIGURE 17: READ RANGE VS. TAG SIZE FOR PROXIMITY APPLICATIONS

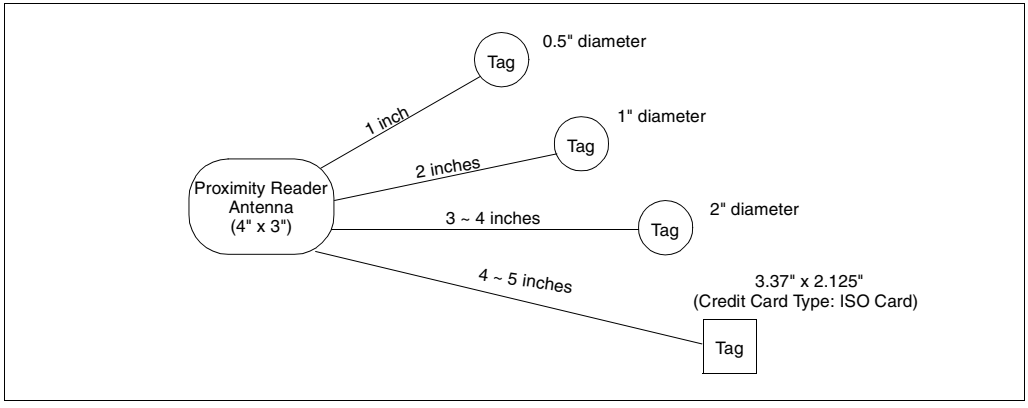
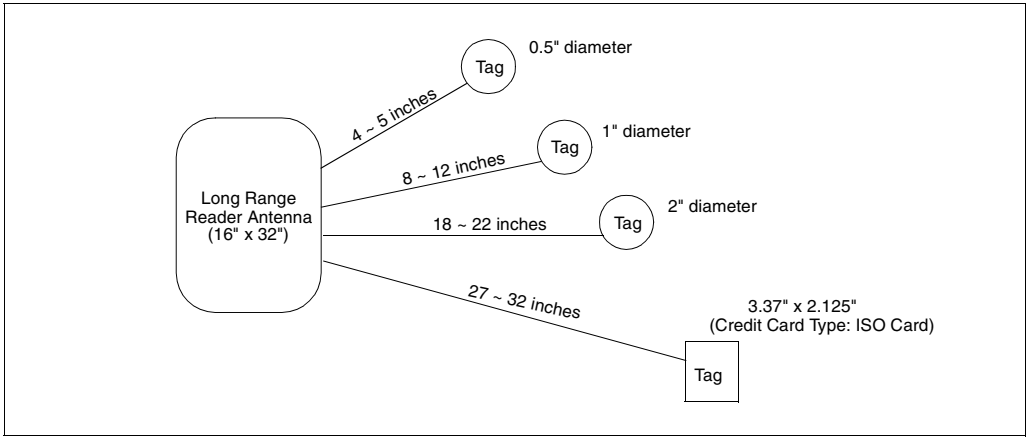


FIGURE 18: READ RANGE VS. TAG SIZE FOR LONG RANGE APPLICATIONS



REFERENCES

1. Frederick W. Grover, Inductance Calculations: Working Formulas and Tables, Dover Publications, Inc., New York, NY., 1946.
2. Keith Henry, Editor, Radio Engineering Handbook, McGraw-Hill Book Company, New York, NY., 1963.
3. V. G. Welsby, The Theory and Design of Inductance Coils, John Wiley and Sons, Inc., 1960.
4. James K. Hardy, High Frequency Circuit Design, Reston Publishing Company, Inc., Reston, Virginia, 1975.

AN680

Passive RFID Basics

Author: *Pete Sorrells*
Microchip Technology Inc.

INTRODUCTION

Radio Frequency Identification (RFID) systems use radio frequency to identify, locate and track people, assets, and animals. Passive RFID systems are composed of three components – an interrogator (reader), a passive tag, and a host computer. The tag is composed of an antenna coil and a silicon chip that includes basic modulation circuitry and non-volatile memory. The tag is energized by a time-varying electromagnetic radio frequency (RF) wave that is transmitted by the reader. This RF signal is called a *carrier signal*. When the RF field passes through an antenna coil, there is an AC voltage generated across the coil. This voltage is rectified to supply power to the tag. The information stored in the tag is transmitted back to the reader. This is often called backscattering. By detecting the backscattering signal, the information stored in the tag can be fully identified.

DEFINITIONS

Reader

Usually a microcontroller-based unit with a wound output coil, peak detector hardware, comparators, and firmware designed to transmit energy to a tag and read information back from it by detecting the backscatter modulation.

Tag

An RFID device incorporating a silicon memory chip (usually with on-board rectification bridge and other RF front-end devices), a wound or printed input/output coil, and (at lower frequencies) a tuning capacitor.

Carrier

A Radio Frequency (RF) sine wave generated by the reader to transmit energy to the tag and retrieve data from the tag. In these examples the ISO frequencies of 125 kHz and 13.56 MHz are assumed; higher frequencies are used for RFID tagging, but the communication methods are somewhat different. 2.45 GHz, for example, uses a true RF link. 125 kHz and 13.56 MHz, utilize transformer-type electromagnetic coupling.

Modulation

Periodic fluctuations in the amplitude of the carrier used to transmit data back from the tag to the reader.

Systems incorporating passive RFID tags operate in ways that may seem unusual to anyone who already understands RF or microwave systems. There is only one transmitter – the passive tag is not a transmitter or transponder in the purest definition of the term, yet bidirectional communication is taking place. The RF field generated by a tag reader (the energy transmitter) has three purposes:

1. **Induce enough power into the tag coil to energize the tag.** Passive tags have no battery or other power source; they must derive all power for operation from the reader field. 125 kHz and 13.56 MHz tag designs must operate over a vast dynamic range of carrier input, from the very near field (in the range of 200 VPP) to the maximum read distance (in the range of 5 VPP).
2. **Provide a synchronized clock source to the tag.** Many RFID tags divide the carrier frequency down to generate an on-board clock for state machines, counters, etc., and to derive the data transmission bit rate for data returned to the reader. Some tags, however, employ on-board oscillators for clock generation.
3. **Act as a carrier for return data from the tag.** Backscatter modulation requires the reader to peak-detect the tag's modulation of the reader's own carrier. See page 20 for additional information on backscatter modulation.

SYSTEM HANDSHAKE

Typical handshake of a tag and reader is as follows:

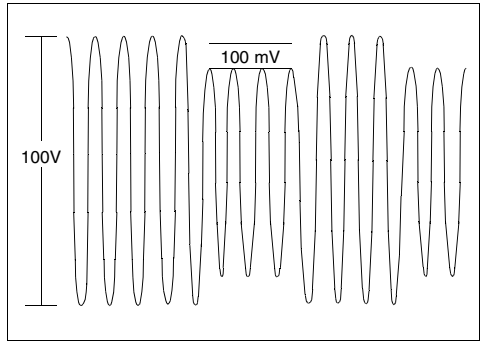
1. The reader continuously generates an RF carrier sine wave, watching always for modulation to occur. Detected modulation of the field would indicate the presence of a tag.
2. A tag enters the RF field generated by the reader. Once the tag has received sufficient energy to operate correctly, it divides down the carrier and begins clocking its data to an output transistor, which is normally connected across the coil inputs.
3. The tag's output transistor shunts the coil, sequentially corresponding to the data which is being clocked out of the memory array.
4. Shunting the coil causes a momentary fluctuation (dampening) of the carrier wave, which is seen as a slight change in amplitude of the carrier.
5. The reader peak-detects the amplitude-modulated data and processes the resulting bitstream according to the encoding and data modulation methods used.

BACKSCATTER MODULATION

This terminology refers to the communication method used by a passive RFID tag to send data back to the reader. By repeatedly shunting the tag coil through a transistor, the tag can cause slight fluctuations in the reader's RF carrier amplitude. The RF link behaves essentially as a transformer; as the secondary winding (tag coil) is momentarily shunted, the primary winding (reader coil) experiences a momentary voltage drop. The reader must peak-detect this data at about 60 dB down (about 100 mV riding on a 100V sine wave) as shown in Figure 1.

This amplitude-modulation loading of the reader's transmitted field provides a communication path back to the reader. The data bits can then be encoded or further modulated in a number of ways.

FIGURE 1: AMPLITUDE – MODULATED BACKSCATTERING SIGNAL

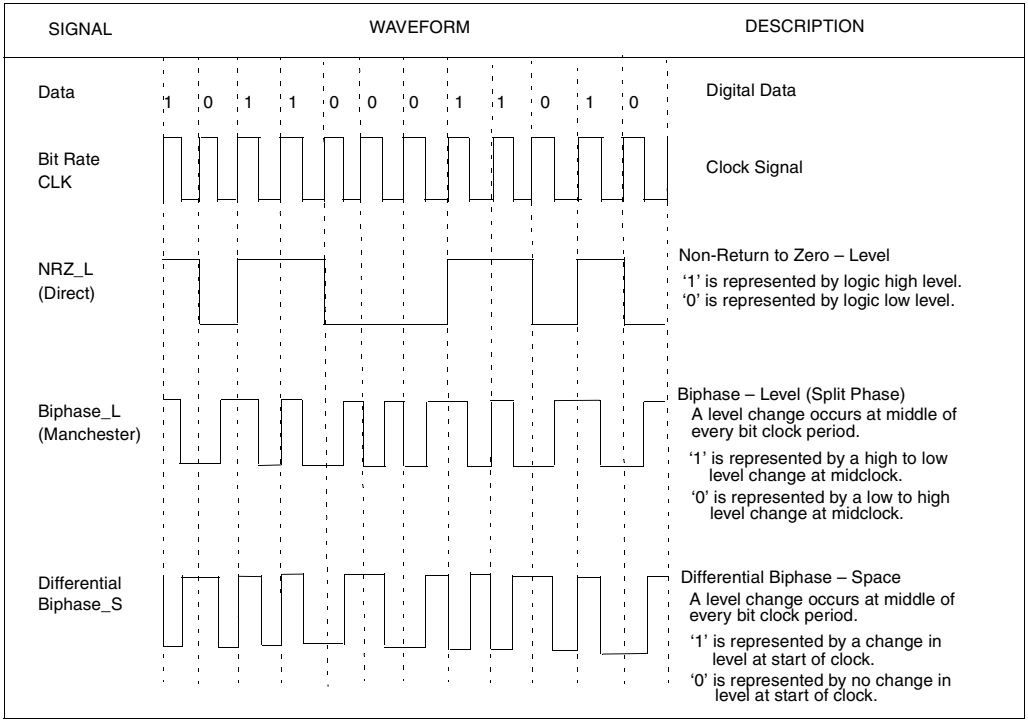


DATA ENCODING

Data encoding refers to processing or altering the data bitstream in-between the time it is retrieved from the RFID chip's data array and its transmission back to the reader. The various encoding algorithms affect error recovery, cost of implementation, bandwidth, synchronization capability, and other aspects of the system design. Entire textbooks are written on the subject, but there are several popular methods used in RFID tagging today:

1. **NRZ (Non-Return to Zero) Direct.** In this method no data encoding is done at all; the 1's and 0's are clocked from the data array directly to the output transistor. A low in the peak-detected modulation is a '0' and a high is a '1'.
2. **Differential Biphase.** Several different forms of differential biphase are used, but in general the bitstream being clocked out of the data array is modified so that a transition always occurs on every clock edge, and 1's and 0's are distinguished by the transitions within the middle of the clock period. This method is used to embed clocking information to help synchronize the reader to the bitstream; and because it always has a transition at a clock edge, it inherently provides some error correction capability. Any clock edge that does not contain a transition in the data stream is in error and can be used to reconstruct the data.
3. **Biphase_L (Manchester).** This is a variation of biphase encoding in which there is not always a transition at the clock edge.

FIGURE 2: VARIOUS DATA CODING WAVEFORMS



DATA MODULATION

Although all the data is transferred to the host by amplitude-modulating the carrier (backscatter modulation), the actual modulation of 1's and 0's is accomplished with three additional modulation methods:

1. **Direct.** In direct modulation, the Amplitude Modulation of the backscatter approach is the only modulation used. A high in the envelope is a '1' and a low is a '0'. Direct modulation can provide a high data rate but low noise immunity.
2. **FSK (Frequency Shift Keying).** This form of modulation uses two different frequencies for data transfer; the most common FSK mode is Fc/8/10. In other words, a '0' is transmitted as an amplitude-modulated clock cycle with period corresponding to the carrier frequency divided by 8, and a '1' is transmitted as an amplitude-modulated clock cycle period corresponding to the carrier frequency divided by 10. The amplitude modulation of the carrier thus switches from Fc/8 to Fc/10 corresponding to 0's and 1's in the bitstream, and the reader has only to count cycles between the peak-detected clock edges to decode the data. FSK allows for a simple reader design, provides very strong noise immunity, but suffers from a lower data rate than some other forms of data modulation. In Figure 3, FSK data modulation is used with NRZ encoding.
3. **PSK (Phase Shift Keying).** This method of data modulation is similar to FSK, except only one frequency is used, and the shift between 1's and 0's is accomplished by shifting the phase of the backscatter clock by 180 degrees. Two common types of PSK are:
   - Change phase at any '0', or
   - Change phase at any data change (0 to 1 or 1 to 0).

   PSK provides fairly good noise immunity, a moderately simple reader design, and a faster data rate than FSK. Typical applications utilize a backscatter clock of Fc/2, as shown in Figure 4.

FIGURE 3: FSK MODULATED SIGNAL, FC/8 = 0, FC/10 = 1

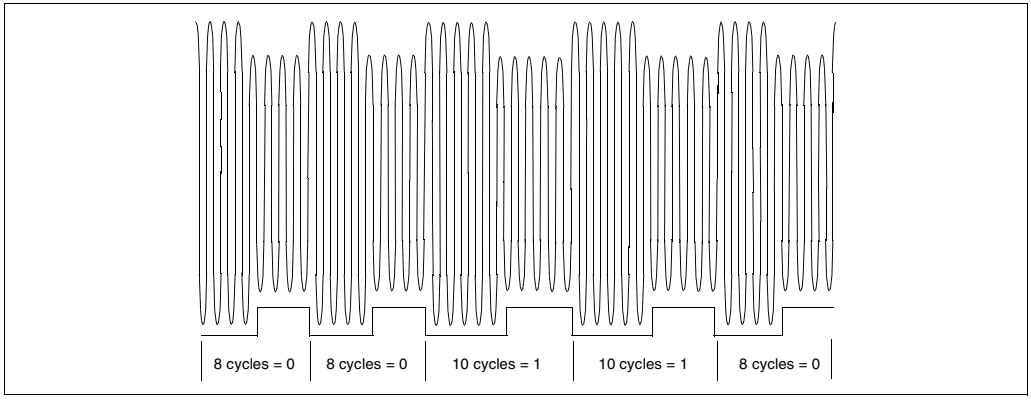
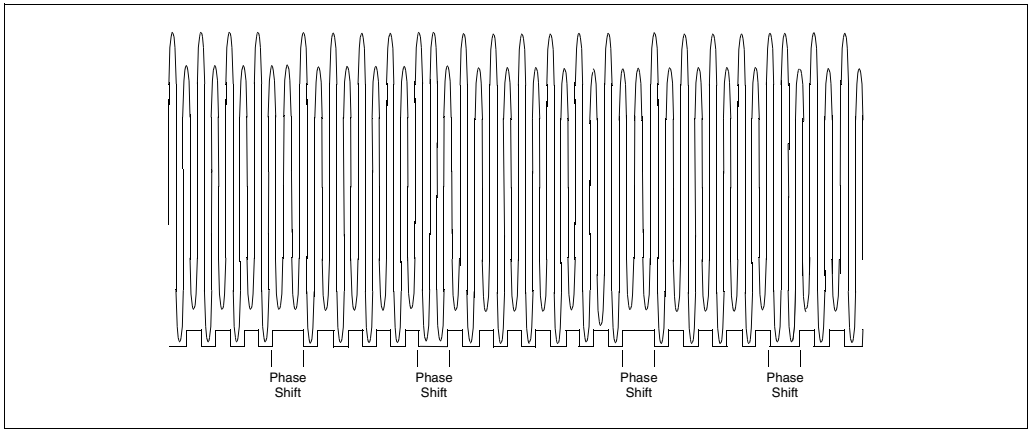


FIGURE 4: PSK MODULATED SIGNAL



ANTICOLLISION

In many existing applications, a single-read RFID tag is sufficient and even necessary: animal tagging and access control are examples. However, in a growing number of new applications, the simultaneous reading of several tags in the same RF field is absolutely critical: library books, airline baggage, garment, and retail applications are a few.

In order to read multiple tags simultaneously, the tag and reader must be designed to detect the condition that more than one tag is active. Otherwise, the tags will all backscatter the carrier at the same time, and the amplitude-modulated waveforms shown in Figures 3 and 4 would be garbled. This is referred to as a *collision*. No data would be transferred to the reader. The tag/reader interface is similar to a serial bus, even though the "bus" travels through the air. In a wired serial bus application, arbitration is necessary to prevent bus contention. The RFID interface also requires arbitration so that only one tag transmits data over the "bus" at one time.

A number of different methods are in use and in development today for preventing collisions; most are patented or patent pending, but all are related to making sure that only one tag "talks" (backscatters) at any one time. See the *MCRF355/360 Data Sheet* (page 7) and the *13.56 MHz Reader Reference Design* (page 47) chapters for more information regarding the MCRF355/360 anticollision protocol.

NOTES:

MCRF 355/360 Applications

Author: *Dr. Youbok Lee, Ph.D.*
Microchip Technology Inc.

INTRODUCTION

The MCRF355 passive RFID device is designed for low cost, multiple reading, and various high volume tagging applications using a frequency band of 13.56 MHz. The device has a total of 154 memory bits that can be reprogrammed by a contact programmer. The device operates with a 70 kHz data rate, and asynchronously with respect to the reader's carrier. The device turns on when the coil voltage reaches 4 VPP and outputs data with a Manchester format (see Figure 2-3 in the data sheet). With the given data rate (70 kHz), it takes about 2.2 ms to transmit all 154 bits of the data. After transmitting all data, the device goes into a sleep mode for 100 ms +/- 50%.

The MCRF355 needs only an external parallel LC resonant circuit that consists of an antenna coil and a capacitor for operation. The external LC components must be connected between antenna A, B, and ground pads. The circuit formed between Antenna Pad A and the ground pad must be tuned to the operating frequency of the reader antenna.

MODE OF OPERATION

The device transmits data by tuning and detuning the resonant frequency of the external circuit. This process is accomplished by using an internal modulation gate (CMOS), that has a very low turn-on resistance (2 ~ 4 ohms) between Drain and Source. This gate turns on during a logic "High" period of the modulation signal and off otherwise. When the gate turns on, its low turn-on resistance shorts the external circuit between Antenna Pad B and the ground pad. Therefore, the resonant frequency of the circuit changes. This is called *detuned* or *cloaking*. Since the detuned tag is out of the frequency band of the reader, the reader can't see it.

The modulation gate turns off as the modulation signal goes to a logic "Low." This turn-off condition again tunes the resonant circuit to the frequency of the reader antenna. Therefore the reader sees the tag again. This is called *tuned* or *uncloaking*.

The tag coil induces maximum voltage during "uncloaking (tuned)" and minimum voltage during cloaking (detuned). Therefore, the cloaking and uncloaking events develop an amplitude modulation signal in the tag coil.

This amplitude modulated signal in the tag coil perturbs the voltage envelope in the reader coil. The reader coil has maximum voltage during cloaking (detuned) and minimum voltage during uncloaking (tuned). By detecting the voltage envelope, the data signal from the tag can be readily reconstructed.

Once the device transmits all 154 bits of data, it goes into "sleep mode" for about 100 ms. The tag wakes up from sleep time (100 ms) and transmits the data package for 2.2 ms and goes into sleep mode again. The device repeats the transmitting and sleep cycles as long as it is energized.

FIGURE 1: VOLTAGE ENVELOPE IN READER COIL

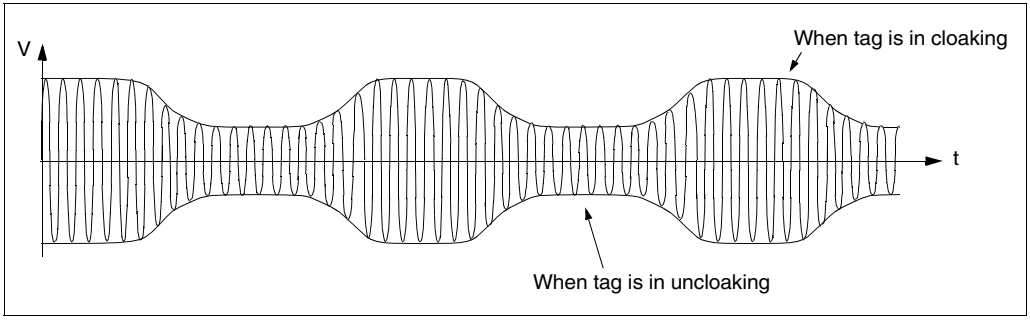
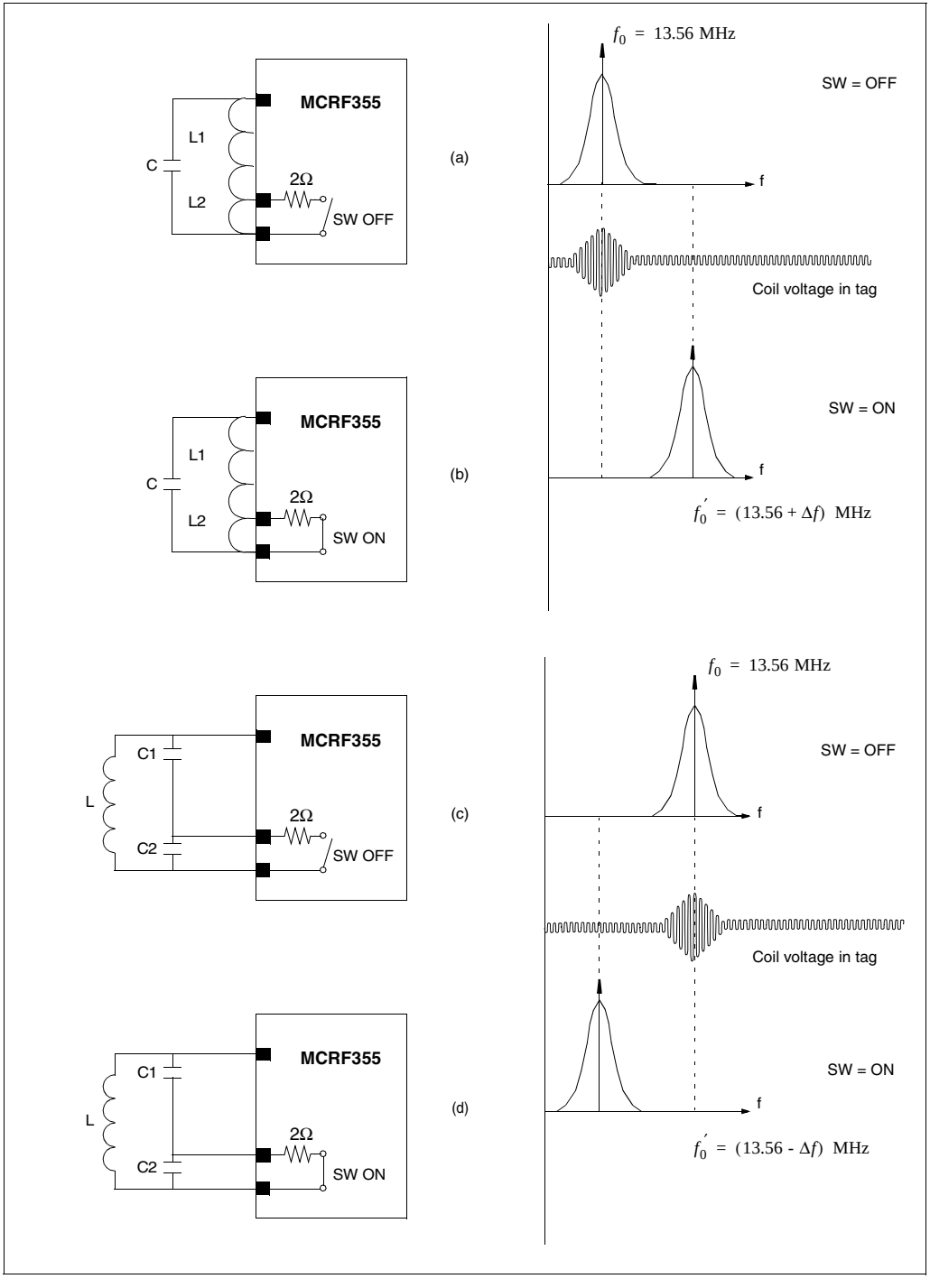


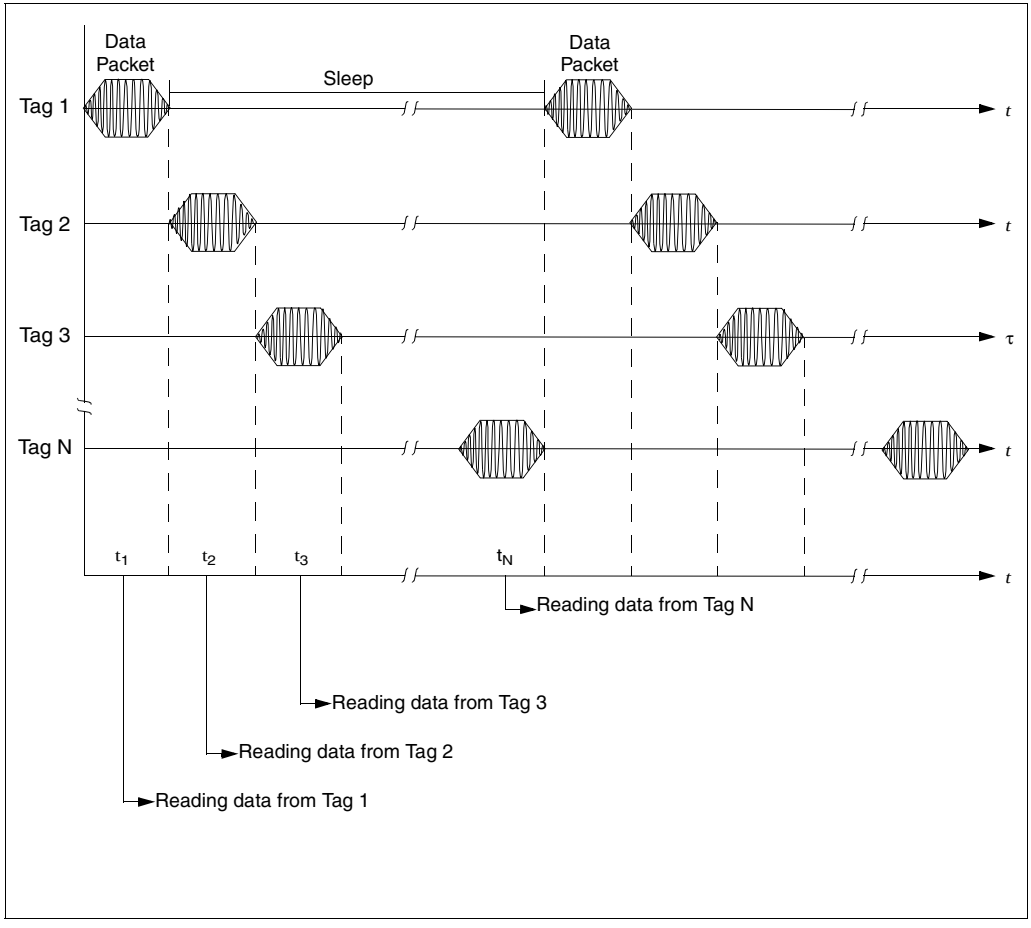
FIGURE 2: (A) UNCLOAKING (TUNED) AND (B) CLOAKING (DETUNED) MODES AND THEIR RESONANT FREQUENCIES



ANTICOLLISION FEATURES

During sleep mode, the device remains in a cloaked state where the circuit is detuned. Therefore, the reader can't see the tag during sleep time. While one tag is in sleep mode, the reader can receive data from other tags. This enables the reader to receive clean data from many tags without any data collision. This ability to read multiple tags in the same RF field is called a*nticollision.* Theoretically, more than 50 tags can be read in the same RF field. However, it is affected by distance from the tag to the reader, angular orientation, movement of the tags, and spacial distribution of the tags.

FIGURE 3: EXAMPLE OF READING MULTIPLE TAGS



EXTERNAL CIRCUIT CONFIGURATION

Since the device transmits data by tuning and detuning the antenna circuit, caution must be given in the external circuit configuration. For a better modulation index, the differences between the tuned and detuned frequencies must be wide enough (about 3 ~ 6 MHz).

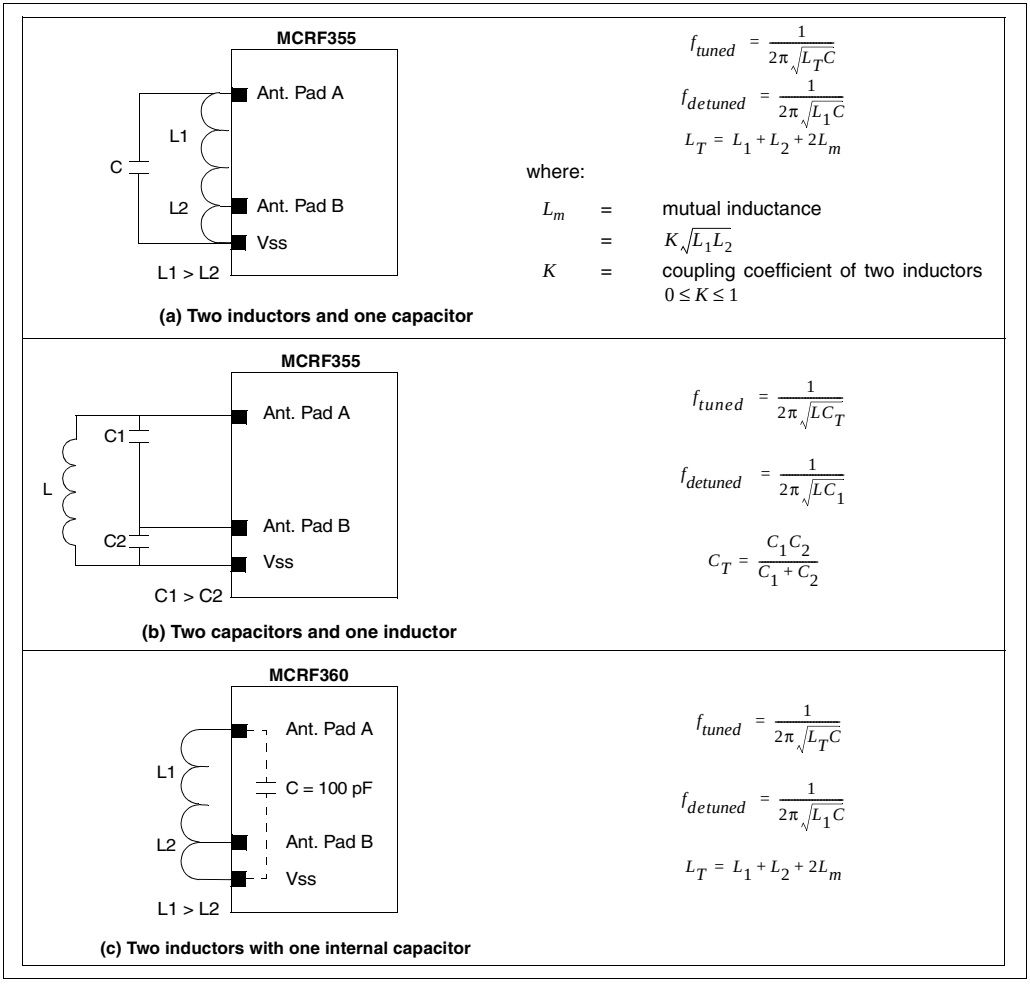
Figure 4 shows various configurations of the external circuit. The choice of the configuration must be chosen depending on the form-factor of the tag. For example, (a) is a better choice for printed circuit tags while, (b) is a better candidate for coil-wound tags. Both (a) and (b) relate to the MCRF355.

In configuration (a), the tuned resonance frequency is determined by a total capacitance and inductance from Antenna Pad A to VSS. During cloaking, the internal switch (modulation gate) shorts Antenna Pad B and VSS. Therefore, the inductance L2 is shorted out. As a result, the detuned frequency is determined by the total capacitance and inductance L1. When shorting the inductance between Antenna Pad B and VSS, the detuned (cloak) frequency is higher than the tuned (uncloak) frequency

In configuration (b), the tuned frequency (uncloak) is determined by the inductance L and the total capacitance between Antenna Pad A and VSS. The circuit detunes (cloak) when C_2 is shorted. This detuned frequency (cloak) is lower than the tuned (uncloak) frequency

The MCRF360 includes a 100 pF internal capacitor. This device needs only an external inductor for operation. The explanation on tuning and detuning is the same as for configuration (a).

FIGURE 4: VARIOUS EXTERNAL CIRCUIT CONFIGURATIONS



$$f_{tuned} = \frac{1}{2\pi\sqrt{L_T C}}$$

$$f_{detuned} = \frac{1}{2\pi\sqrt{L_1 C}}$$

$$L_T = L_1 + L_2 + 2L_m$$

where:

L_m = mutual inductance

$= K\sqrt{L_1 L_2}$

K = coupling coefficient of two inductors $0 \le K \le 1$

$$f_{tuned} = \frac{1}{2\pi\sqrt{LC_T}}$$

$$f_{detuned} = \frac{1}{2\pi\sqrt{LC_1}}$$

$$C_T = \frac{C_1 C_2}{C_1 + C_2}$$

$$f_{tuned} = \frac{1}{2\pi\sqrt{L_T C}}$$

$$f_{detuned} = \frac{1}{2\pi\sqrt{L_1 C}}$$

$$L_T = L_1 + L_2 + 2L_m$$

PROGRAMMING OF DEVICE

All of the memory bits in the MCRF355/360 are reprogrammable by a contact programmer or by factory programming prior to shipment, known as Serialized Quick Turn Programming^{SM} (SQTP^{SM}). For more information about contact programming, see page 69 of the *microID™ 13.56 MHz System Design Guide* (DS21299). For information about SQTP programming, please see TB032 (DS91032), page 19 of the design guide.

Serial Quick Turn Programming (SQTP) is a Service Mark of Microchip Technology Inc.

NOTES:

Antenna Circuit Design

Author: *Dr. Youbok Lee, Ph.D.*
Microchip Technology Inc.

INTRODUCTION

Passive RFID tags utilize an induced antenna coil voltage for operation. This induced AC voltage is rectified to provide a voltage source for the device. As the DC voltage reaches a certain level, the device starts operating. By providing an energizing RF signal, a reader can communicate with a remotely located device that has no external power source such as a battery. Since the energizing and communication between the reader and tag is accomplished through antenna coils, it is important that the device must be equipped with a proper antenna circuit for successful RFID applications.

An RF signal can be radiated effectively if the linear dimension of the antenna is comparable with the wavelength of the operating frequency. However, the wavelength at 13.56 MHz is 22.12 meters. Therefore, it is difficult to form a true antenna for most RFID applications. Alternatively, a small loop antenna circuit that is resonating at the frequency is used. A current flowing into the coil radiates a near-field magnetic field that falls off with r^{-3}. This type of antenna is called a *magnetic dipole antenna.*

For 13.56 MHz passive tag applications, a few microhenries of inductance and a few hundred pF of resonant capacitor are typically used. The voltage transfer between the reader and tag coils is accomplished through inductive coupling between the two coils. As in a typical transformer, where a voltage in the primary coil transfers to the secondary coil, the voltage in the reader antenna coil is transferred to the tag antenna coil and vice versa. The efficiency of the voltage transfer can be increased significantly with high *Q* circuits.

This section is written for RF coil designers and RFID system engineers. It reviews basic electromagnetic theories on antenna coils, a procedure for coil design, calculation and measurement of inductance, an antenna tuning method, and read range in RFID applications.

REVIEW OF A BASIC THEORY FOR RFID ANTENNA DESIGN

Current and Magnetic Fields

Ampere's law states that current flowing in a conductor produces a magnetic field around the conductor. The magnetic field produced by a current element, as shown in Figure 1, on a round conductor (wire) with a finite length is given by:

EQUATION 1:

$$B_\phi = \frac{\mu_o I}{4\pi r}(\cos\alpha_2 - \cos\alpha_1) \qquad (\text{Weber}/m^2)$$

where:

I = current
r = distance from the center of wire
μ_0 = permeability of free space and given as $4\pi \times 10^{-7}$ (Henry/meter)

In a special case with an infinitely long wire where:

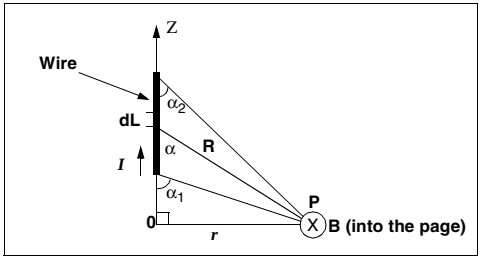
α_1 = -180°
α_2 = 0°

Equation 1 can be rewritten as:

EQUATION 2:

$$B_\phi = \frac{\mu_o I}{2\pi r} \qquad (\text{Weber}/m^2)$$

FIGURE 1: CALCULATION OF MAGNETIC FIELD B AT LOCATION P DUE TO CURRENT I ON A STRAIGHT CONDUCTING WIRE



The magnetic field produced by a circular loop antenna is given by:

EQUATION 3:

$$B_z = \frac{\mu_o I N a^2}{2(a^2 + r^2)^{3/2}}$$

$$= \frac{\mu_o I N a^2}{2} \left(\frac{1}{r^3}\right) \quad \text{for} \quad r^2 >> a^2$$

where

I	=	current
a	=	radius of loop
r	=	distance from the center of wire
μ_0	=	permeability of free space and given as $\mu_o = 4\pi \times 10^{-7}$ (Henry/meter)

The above equation indicates that the magnetic field strength decays with $1/r^3$. A graphical demonstration is shown in Figure 3. It has maximum amplitude in the plane of the loop and directly proportional to both the current and the number of turns, N.

Equation 3 is often used to calculate the ampere-turn requirement for read range. A few examples that calculate the ampere-turns and the field intensity necessary to power the tag will be given in the following sections.

FIGURE 2: CALCULATION OF MAGNETIC FIELD B AT LOCATION P DUE TO CURRENT I ON THE LOOP

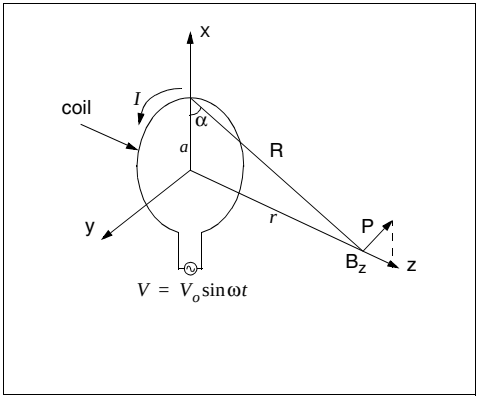
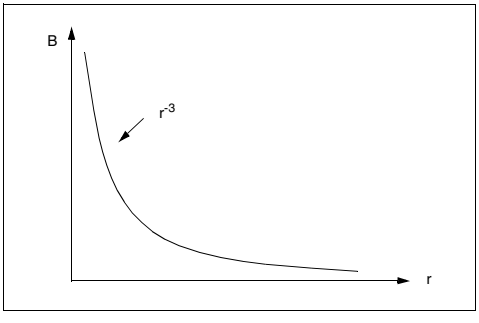


FIGURE 3: DECAYING OF THE MAGNETIC FIELD B VS. DISTANCE *r*



INDUCED VOLTAGE IN AN ANTENNA COIL

Faraday's law states that a time-varying magnetic field through a surface bounded by a closed path induces a voltage around the loop.

Figure 4 shows a simple geometry of an RFID application. When the tag and reader antennas are in close proximity, the time-varying magnetic field B that is produced by a reader antenna coil induces a voltage (called electromotive force or simply EMF) in the closed tag antenna coil. The induced voltage in the coil causes a flow of current on the coil. This is called Faraday's law. The induced voltage on the tag antenna coil is equal to the time rate of change of the magnetic flux Ψ.

EQUATION 4:

$$V = -N\frac{d\psi}{dt}$$

where:

N	=	number of turns in the antenna coil
Ψ	=	magnetic flux through each turn

The negative sign shows that the induced voltage acts in such a way as to oppose the magnetic flux producing it. This is known as Lenz's Law and it emphasizes the fact that the direction of current flow in the circuit is such that the induced magnetic field produced by the induced current will oppose the original magnetic field.

The magnetic flux Ψ in Equation 4 is the total magnetic field B that is passing through the entire surface of the antenna coil, and found by:

EQUATION 5:

$$\psi = \int B \cdot dS$$

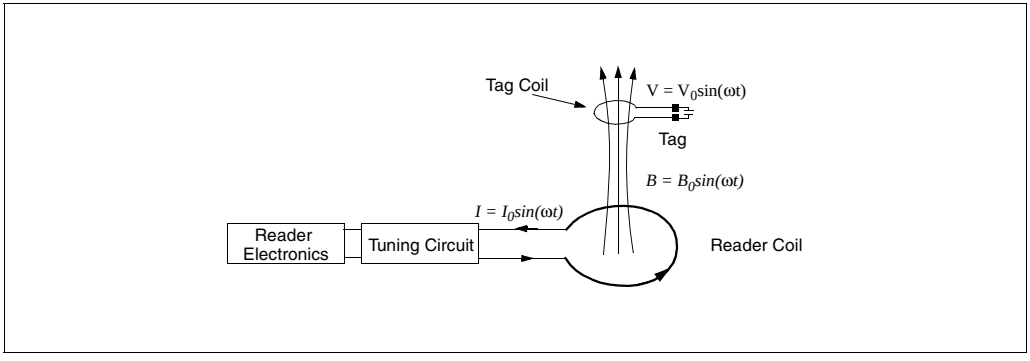
where:

B	=	magnetic field given in Equation 2
S	=	surface area of the coil
•	=	inner product (*cosine angle between two vectors*) of vectors B and surface area S

Note:	Both magnetic field B and surface S are vector quantities.

The presentation of inner product of two vectors in Equation 5 suggests that the total magnetic flux ψ that is passing through the antenna coil is affected by an orientation of the antenna coils. The inner product of two vectors becomes maximized when the cosine angle between the two are 90 degree, or the two (B field and the surface of coil) are perpendicular to each other. The maximum magnetic flux that is passing through the tag coil is obtained when the two coils (reader coil and tag coil) are placed in parallel with respect to each other. This condition results in maximum induced voltage in the tag coil and also maximum read range. The inner product expression in Equation 5 also can be expressed in terms of a mutual coupling between the reader and tag coils. The mutual coupling between the two coils is maximized in the above condition.

FIGURE 4: A BASIC CONFIGURATION OF READER AND TAG ANTENNAS IN RFID APPLICATIONS



Using Equations 3 and 5, Equation 4 can be rewritten as:

EQUATION 6:

$$V = -N_2 \frac{d\Psi_{21}}{dt} = -N_2 \frac{d}{dt}\left(\int B \cdot dS\right)$$

$$= -N_2 \frac{d}{dt}\left[\int \frac{\mu_o i_1 N_1 a^2}{2(a^2 + r^2)^{3/2}} \cdot dS\right]$$

$$= -\left[\frac{\mu_o N_1 N_2 a^2 (\pi b^2)}{2(a^2 + r^2)^{3/2}}\right]\frac{di_1}{dt}$$

$$= -M \frac{di_1}{dt}$$

where:

V	=	voltage in the tag coil
i_1	=	current on the reader coil
a	=	radius of the reader coil
b	=	radius of tag coil
r	=	distance between the two coils
M	=	mutual inductance between the tag and reader coils, and given by:

EQUATION 7:

$$M = \left[\frac{\mu_o \pi N_1 N_2 (ab)^2}{2(a^2 + r^2)^{3/2}}\right]$$

The above equation is equivalent to a voltage transformation in typical transformer applications. The current flow in the primary coil produces a magnetic flux that causes a voltage induction at the secondary coil.

As shown in Equation 6, the tag coil voltage is largely dependent on the mutual inductance between the two coils. The mutual inductance is a function of coil geometry and the spacing between them. The induced voltage in the tag coil decreases with r^{-3}. Therefore, the read range also decreases in the same way.

From Equations 4 and 5, a generalized expression for induced voltage V_o in a tuned loop coil is given by:

EQUATION 8:

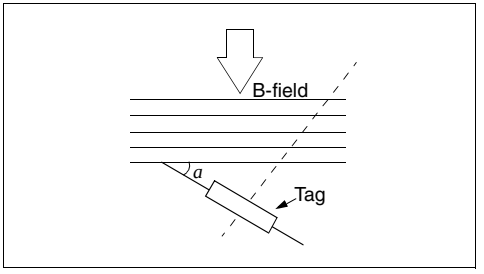
$$V_0 = 2\pi f N S Q B_o \cos\alpha$$

where:

f	=	frequency of the arrival signal
N	=	number of turns of coil in the loop
S	=	area of the loop in square meters (m^2)
Q	=	quality factor of circuit
B_o	=	strength of the arrival signal
α	=	angle of arrival of the signal

In the above equation, the quality factor Q is a measure of the selectivity of the frequency of the interest. The Q will be defined in Equations 31 through 47.

FIGURE 5: ORIENTATION DEPENDENCY OF THE TAG ANTENNA



The induced voltage developed across the loop antenna coil is a function of the angle of the arrival signal. The induced voltage is maximized when the antenna coil is placed in parallel with the incoming signal where $\alpha = 0$.

EXAMPLE 1: CALCULATION OF B-FIELD IN A TAG COIL

The MCRF355 device turns on when the antenna coil develops 4 VPP across it. This voltage is rectified and the device starts to operate when it reaches 2.4 VDC. The B-field to induce a 4 VPP coil voltage with an ISO standard 7810 card size (85.6 x 54 x 0.76 mm) is calculated from the coil voltage equation using Equation 8.

EQUATION 9:

$$V_o = 2\pi f N S Q B_o \cos\alpha = 4$$

and

$$B_o = \frac{4/(\sqrt{2})}{2\pi f N S Q \cos\alpha} = 0.0449 \qquad (\mu wbm^{-2})$$

where the following parameters are used in the above calculation:

Tag coil size	=	(85.6 x 54) mm^2 (ISO card size) = 0.0046224 m^2
Frequency	=	13.56 MHz
Number of turns	=	4
Q of tag antenna coil	=	40
AC coil voltage to turn on the tag	=	4 VPP
$\cos\alpha$ =	=	1 (normal direction, $\alpha = 0$).

EXAMPLE 2: NUMBER OF TURNS AND CURRENT (AMPERE-TURNS)

Assuming that the reader should provide a read range of 15 inches (38.1 cm) for the tag given in the previous example, the current and number of turns of a reader antenna coil is calculated from Equation 3:

EQUATION 10:

$$(NI)_{rms} = \frac{2B_z(a^2 + r^2)^{3/2}}{\mu a^2}$$

$$= \frac{2(0.0449 \times 10^{-6})(0.1^2 + (0.38)^2)^{3/2}}{(4\pi \times 10^{-7})(0.1^2)}$$

$$= 0.43(\text{ampere - turns})$$

The above result indicates that it needs a 430 mA for 1 turn coil, and 215 mA for 2-turn coil.

EXAMPLE 3: OPTIMUM COIL DIAMETER OF THE READER COIL

An optimum coil diameter that requires the minimum number of ampere-turns for a particular read range can be found from Equation 3 such as:

EQUATION 11:

$$NI = K\frac{(a^2 + r^2)^{\frac{3}{2}}}{a^2}$$

where: $K = \frac{2B_z}{\mu_o}$

By taking derivative with respect to the radius a,

$$\frac{d(NI)}{da} = K\frac{3/2(a^2 + r^2)^{1/2}(2a^3) - 2a(a^2 + r^2)^{3/2}}{a^4}$$

$$= K\frac{(a^2 - 2r^2)(a^2 + r^2)^{1/2}}{a^3}$$

The above equation becomes minimized when:

$$a^2 - 2r^2 = 0$$

The above result shows a relationship between the read range vs. optimum coil diameter. The optimum coil diameter is found as:

EQUATION 12:

$$a = \sqrt{2}r$$

where:

a = radius of coil

r = read range.

The result indicates that the optimum loop radius, a, is 1.414 times the demanded read range r.

WIRE TYPES AND OHMIC LOSSES

Wire Size and DC Resistance

The diameter of electrical wire is expressed as the American Wire Gauge (AWG) number. The gauge number is inversely proportional to diameter, and the diameter is roughly doubled every six wire gauges. The wire with a smaller diameter has a higher DC resistance. The DC resistance for a conductor with a uniform cross-sectional area is found by:

EQUATION 13:

$$R_{DC} = \frac{l}{\sigma S} \qquad (\Omega)$$

where:

l = total length of the wire
σ = conductivity
S = cross-sectional area

Table 1 shows the diameter for bare and enamel-coated wires, and DC resistance.

AC Resistance of Wire

At DC, charge carriers are evenly distributed through the entire cross section of a wire. As the frequency increases, the reactance near the center of the wire increases. This results in higher impedance to the current density in the region. Therefore, the charge moves away from the center of the wire and towards the edge of the wire. As a result, the current density decreases in the center of the wire and increases near the edge of the wire. This is called a *skin effect*. The depth into the conductor at which the current density falls to 1/e, or 37% of its value along the surface, is known as the *skin depth* and is a function of the frequency and the permeability and conductivity of the medium. The skin depth is given by:

EQUATION 14:

$$\delta = \frac{1}{\sqrt{\pi f \mu \sigma}}$$

where:

f = frequency
μ = permeability of material
σ = conductivity of the material

EXAMPLE 4:

The skin depth for a copper wire at 13.56 MHz can be calculated as:

EQUATION 15:

$$\delta = \frac{1}{\sqrt{\pi f (4\pi \times 10^{-7})(5.8 \times 10^{-7})}}$$

$$= \frac{0.0179}{\sqrt{f}} \qquad (m)$$

$$= 0.187 \qquad (mm)$$

The wire resistance increases with frequency, and the resistance due to the skin depth is called an AC resistance. An approximated formula for the AC resistance is given by:

EQUATION 16:

$$R_{ac} \approx \frac{1}{2\sigma\pi\delta} = (R_{DC})\frac{a}{2\delta} \qquad (\Omega)$$

where:

a = coil radius

TABLE 1: AWG WIRE CHART

Wire Size (AWG)	Dia. in Mils (bare)	Dia. in Mils (coated)	Ohms/ 1000 ft.	Cross Section (mils)
1	289.3	—	0.126	83690
2	287.6	—	0.156	66360
3	229.4	—	0.197	52620
4	204.3	—	0.249	41740
5	181.9	—	0.313	33090
6	162.0	—	0.395	26240
7	166.3	—	0.498	20820
8	128.5	131.6	0.628	16510
9	114.4	116.3	0.793	13090
10	101.9	106.2	0.999	10380
11	90.7	93.5	1.26	8230
12	80.8	83.3	1.59	6530
13	72.0	74.1	2.00	5180
14	64.1	66.7	2.52	4110
15	57.1	59.5	3.18	3260
16	50.8	52.9	4.02	2580
17	45.3	47.2	5.05	2060
18	40.3	42.4	6.39	1620
19	35.9	37.9	8.05	1290
20	32.0	34.0	10.1	1020
21	28.5	30.2	12.8	812
22	25.3	28.0	16.2	640
23	22.6	24.2	20.3	511
24	20.1	21.6	25.7	404
25	17.9	19.3	32.4	320

Note: 1 mil = 2.54 x 10^{-3} cm

Wire Size (AWG)	Dia. in Mils (bare)	Dia. in Mils (coated)	Ohms/ 1000 ft.	Cross Section (mils)
26	15.9	17.2	41.0	253
27	14.2	15.4	51.4	202
28	12.6	13.8	65.3	159
29	11.3	12.3	81.2	123
30	10.0	11.0	106.0	100
31	8.9	9.9	131	79.2
32	8.0	8.8	162	64.0
33	7.1	7.9	206	50.4
34	6.3	7.0	261	39.7
35	5.6	6.3	331	31.4
36	5.0	5.7	415	25.0
37	4.5	5.1	512	20.2
38	4.0	4.5	648	16.0
39	3.5	4.0	847	12.2
40	3.1	3.5	1080	9.61
41	2.8	3.1	1320	7.84
42	2.5	2.8	1660	6.25
43	2.2	2.5	2140	4.84
44	2.0	2.3	2590	4.00
45	1.76	1.9	3350	3.10
46	1.57	1.7	4210	2.46
47	1.40	1.6	5290	1.96
48	1.24	1.4	6750	1.54
49	1.11	1.3	8420	1.23
50	0.99	1.1	10600	0.98

Note: 1 mil = 2.54 x 10^{-3} cm

INDUCTANCE OF VARIOUS ANTENNA COILS

An electric current element that flows through a conductor produces a magnetic field. This time-varying magnetic field is capable of producing a flow of current through another conductor – this is called *inductance*. The inductance L depends on the physical characteristics of the conductor. A coil has more inductance than a straight wire of the same material, and a coil with more turns has more inductance than a coil with fewer turns. The inductance L of inductor is defined as the ratio of the total magnetic flux linkage to the current I through the inductor:

EQUATION 17:

$$L = \frac{N\psi}{I} \qquad \text{(Henry)}$$

where:

N = number of turns
I = current
Ψ = the magnetic flux

For a coil with multiple turns, the inductance is greater as the spacing between turns becomes smaller. Therefore, the tag antenna coil that has to be formed in a limited space often needs a multilayer winding to reduce the number of turns.

Calculation of Inductance

Inductance of the coil can be calculated in many different ways. Some are readily available from references[1-4]. It must be remembered that for RF coils the actual resulting inductance may differ from the calculated true result because of distributed capacitance. For that reason, inductance calculations are generally used only for a starting point in the final design.

Inductance of a Straight Wound Wire

The inductance of a straight wound wire shown in Figure 1 is given by:

EQUATION 18:

$$L = 0.002l\left[\log_e\frac{2l}{a} - \frac{3}{4}\right] \qquad (\mu H)$$

where:

l and a = length and radius of wire in cm, respectively.

EXAMPLE 5: INDUCTANCE CALCULATION FOR A STRAIGHT WIRE:

The inductance of a wire with 10 feet (304.8cm) long and 2 mm in diameter is calculated as follows:

EQUATION 19:

$$L = 0.002(304.8)\left[\ln\left(\frac{2(304.8)}{0.1}\right) - \frac{3}{4}\right]$$

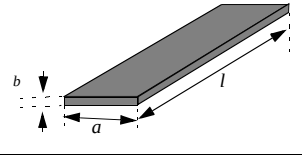
$$= 0.60967(7.965)$$

$$= 4.855(\mu H)$$

Inductance of Thin Film Inductor with a Rectangular Cross Section

Inductance of a conductor with rectangular cross section as shown in Figure 6 is calculated as:

FIGURE 6: A STRAIGHT THIN FILM INDUCTOR



EQUATION 20:

$$L = 0.002l\left\{\ln\left(\frac{2l}{a+b}\right) + 0.50049 + \frac{a+b}{3l}\right\} \qquad (\mu H)$$

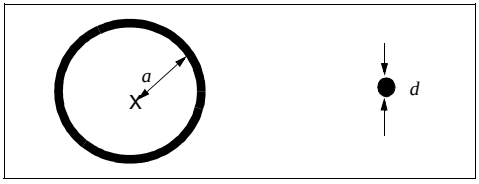
where:

a = width in cm
b = thickness in cm
l = length of conductor in cm

Inductance of a Circular Coil with Single Turn

The inductance of a circular coil shown in Figure 7 can be calculated by:

FIGURE 7: A CIRCULAR COIL WITH SINGLE TURN



EQUATION 21:

$$L = 0.01257(a)\left[2.303\log_{10}\left(\frac{16a}{d} - 2\right)\right] \qquad (\mu H)$$

where:

- a = mean radius of loop in (cm)
- d = diameter of wire in (cm)

Inductance of an N-turn Circular Coil with Single Layer

The inductance of a circular coil with single layer is calculated as:

EQUATION 22:

$$L = \frac{(aN)^2}{22.9l + 25.4a} \qquad (\mu H)$$

where:

- N = number of turns
- l = length
- a = the radius of coil in cm

Inductance of N-turn Circular Coil with Multilayer

FIGURE 8: N-TURN CIRCULAR COIL WITH SINGLE LAYER

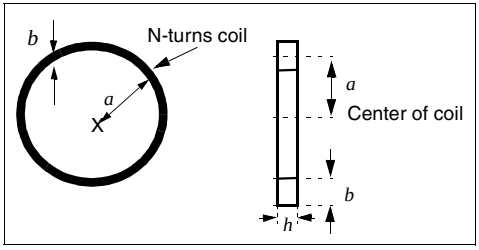


Figure 8 shows an N-turn inductor of circular coil with multilayer. Its inductance is calculated by:

EQUATION 23:

$$L = \frac{0.31(aN)^2}{6a + 9h + 10b} \qquad (\mu H)$$

where:

- a = average radius of the coil in cm
- N = number of turns
- b = winding thickness in cm
- h = winding height in cm

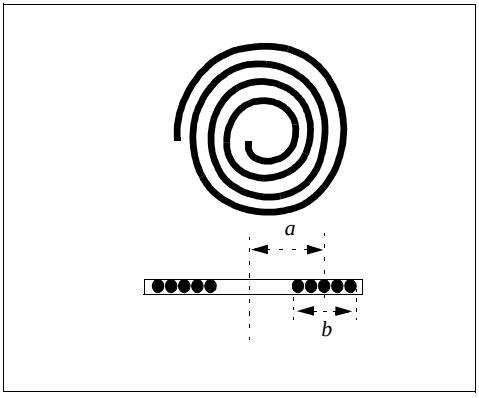
Inductance of Spiral Wound Coil with Single Layer

The inductance of a spiral inductor is calculated by:

EQUATION 24:

$$L = \frac{(aN)^2}{8a + 11b} \qquad (\mu H)$$

FIGURE 9: A SPIRAL COIL



Inductance of N-turn Square Loop Coil with Multilayer

Inductance of a multilayer square loop coil is calculated by:

EQUATION 25:

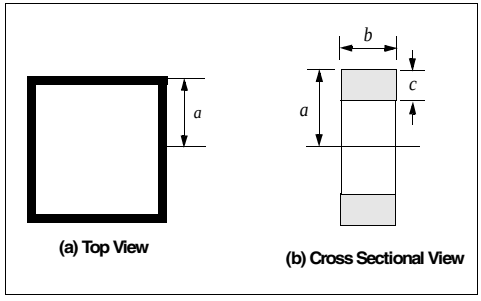
$$L = 0.008aN^2\left\{2.303\log_{10}\left(\frac{a}{b+c}\right) + 0.2235\frac{b+c}{a} + 0.726\right\}(\mu H)$$

where:

N	=	number of turns
a	=	side of square measured to the center of the rectangular cross section of winding
b	=	winding length
c	=	winding depth as shown in Figure 10.

Note: All dimensions are in cm.

FIGURE 10: N-TURN SQUARE LOOP COIL WITH MULTILAYER



Inductance of a Flat Square Coil

Inductance of a flat square coil of rectangular cross section with N turns is calculated by[4]:

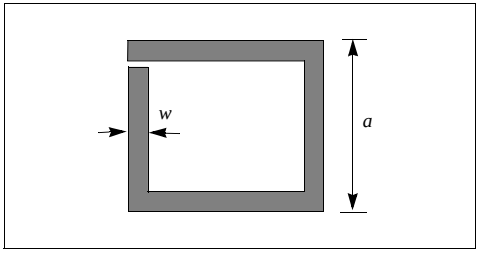
EQUATION 26:

$$= 0.0467aN^2\left\{\log_{10}\left(2\frac{a^2}{t+w}\right) - \log_{10}(2.414a)\right\} + 0.02032aN^2\left\{0.914 + \left[\frac{0.2235}{a}(t+w)\right]\right\}$$

where:

L = in µH
a = side length in inches
t = thickness in inches
w = width in inches

FIGURE 11: SQUARE LOOP INDUCTOR WITH A RECTANGULAR CROSS SECTION



The formulas for inductance are widely published and provide a reasonable approximation for the relationship between inductance and the number of turns for a given physical size[1–4]. When building prototype coils, it is wise to exceed the number of calculated turns by about 10% and then remove turns to achieve a right value. For production coils, it is best to specify an inductance and tolerance rather than a specific number of turns.

CONFIGURATION OF ANTENNA CIRCUITS

Reader Antenna Circuits

The inductance for the reader antenna coil for 13.56 MHz is typically in the range of a few microhenries (μH). The antenna can be formed by aircore or ferrite core inductors. The antenna can also be formed by a metallic or conductive trace on PCB board or on flexible substrate.

The reader antenna can be made of either a single coil, that is typically forming a series or a parallel resonant circuit, or a double loop (transformer) antenna coil. Figure 12 shows various configurations of reader antenna circuit. The coil circuit must be tuned to the operating frequency to maximize power efficiency. The tuned LC resonant circuit is the same as the bandpass filter that passes only a selected frequency. The Q of the tuned circuit is related to both read range and bandwidth of the circuit. More on this subject will be discussed in the following section.

Choosing the size and type of antenna circuit depends on the system design topology. The series resonant circuit results in minimum impedance at the resonance frequency. Therefore, it draws a maximum current at the resonance frequency. Because of its simple circuit topology and relatively low cost, this type of antenna circuit is suitable for proximity reader antenna.

On the other hand, a parallel resonant circuit results in maximum impedance at the resonance frequency. Therefore, maximum voltage is available at the resonance frequency. Although it has a minimum resonant current, it still has a strong circulating current that is proportional to Q of the circuit. The double loop antenna coil that is formed by two parallel antenna circuits can also be used.

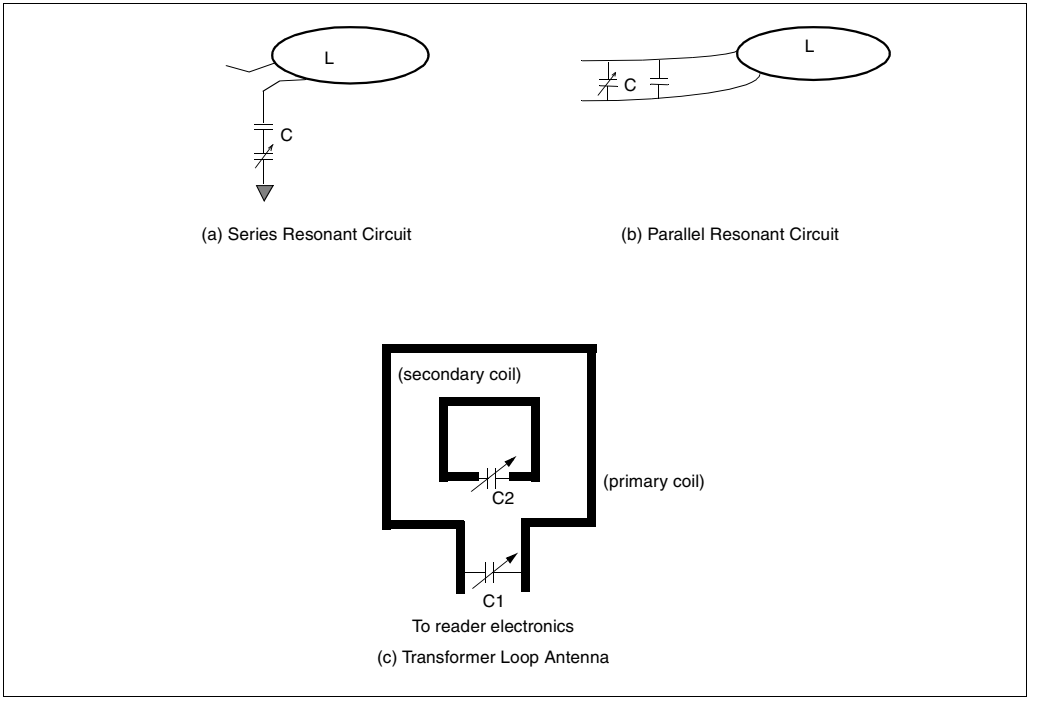
The frequency tolerance of the carrier frequency and output power level from the read antenna is regulated by government regulations (e.g., FCC in the USA).

FCC limits for 13.56 MHz frequency band are as follows:

1. Tolerance of the carrier frequency: 13.56 MHz +/- 0.01% = +/- 1.356 kHz.
2. Frequency bandwidth: +/- 7 kHz.
3. Power level of fundamental frequency: 10 mv/m at 30 meters from the transmitter.
4. Power level for harmonics: -50.45 dB down from the fundamental signal.

The transmission circuit including the antenna coil must be designed to meet the FCC limits.

FIGURE 12: VARIOUS READER ANTENNA CIRCUITS



Tag Antenna Circuits

The MCRF355 device communicates data by tuning and detuning the antenna circuit (see AN707). Figure 13 shows examples of the external circuit arrangement.

The external circuit must be tuned to the resonant frequency of the reader antenna. In a detuned condition, a circuit element between the antenna B and VSS pads is shorted. The frequency difference (delta frequency) between tuned and detuned frequencies must be adjusted properly for optimum operation. It has been found that maximum modulation index and maximum read range occur when the tuned and detuned frequencies are separated by 3 to 6 MHz.

The tuned frequency is formed from the circuit elements between the antenna A and VSS pads without shorting the antenna B pad. The detuned frequency is found when the antenna B pad is shorted. This detuned frequency is calculated from the circuit between antenna A and VSS pads excluding the circuit element between antenna B and VSS pads.

In Figure 13 (a), the tuned resonant frequency is

EQUATION 27:

$$f_o = \frac{1}{2\pi\sqrt{L_T C}}$$

where:

L_T = $L_1 + L_2 + 2L_M$ = Total inductance between antenna A and VSS pads

L_1 = inductance between antenna A and antenna B pads

L_2 = inductance between ant. B and VSS pads

M = mutual inductance between coil 1 and coil 2

= $k\sqrt{L_1 L_2}$

k = coupling coefficient between the two coils

C = tuning capacitance

and detuned frequency is

EQUATION 28:

$$f_{detuned} = \frac{1}{2\pi\sqrt{L_1 C}}$$

In this case, $f_{detuned}$ is higher than f_{tuned}.

Figure 13(b) shows another example of the external circuit arrangement. This configuration controls C_2 for tuned and detuned frequencies. The tuned and untuned frequencies are

EQUATION 29:

$$f_{tuned} = \frac{1}{2\pi\sqrt{\left(\frac{C_1 C_2}{C_1 + C_2}\right)L}}$$

and

EQUATION 30:

$$f_{detuned} = \frac{1}{2\pi\sqrt{LC_1}}$$

A typical inductance of the coil is about a few microhenry with a few turns. Once the inductance is determined, the resonant capacitance is calculated from the above equations. For example, if a coil has an inductance of 1.3 µH, then it needs a 106 pF of capacitance to resonate at 13.56 MHz.

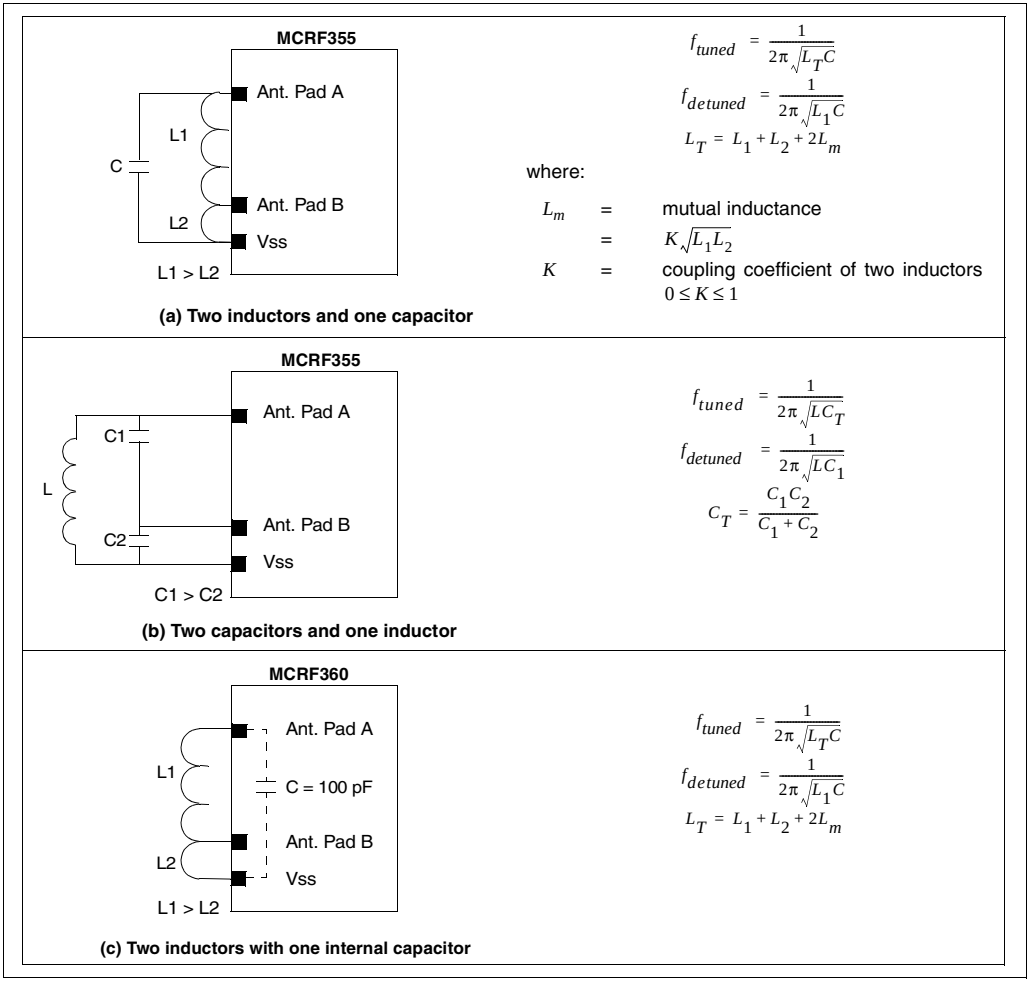
CONSIDERATION ON QUALITY FACTOR *Q* AND BANDWIDTH OF TUNING CIRCUIT

The voltage across the coil is a product of quality factor *Q* of the circuit and input voltage. Therefore, for a given input voltage signal, the coil voltage is directly proportional to the *Q* of the circuit. In general, a higher *Q* results in longer read range. However, the *Q* is also related to the bandwidth of the circuit as shown in the following equation.

EQUATION 31:

$$Q = \frac{f_o}{B}$$

FIGURE 13: VARIOUS EXTERNAL CIRCUIT CONFIGURATIONS



$$f_{tuned} = \frac{1}{2\pi\sqrt{L_T C}}$$

$$f_{detuned} = \frac{1}{2\pi\sqrt{L_1 C}}$$

$$L_T = L_1 + L_2 + 2L_m$$

where:

L_m = mutual inductance

$= K\sqrt{L_1 L_2}$

K = coupling coefficient of two inductors $0 \le K \le 1$

$$f_{tuned} = \frac{1}{2\pi\sqrt{LC_T}}$$

$$f_{detuned} = \frac{1}{2\pi\sqrt{LC_1}}$$

$$C_T = \frac{C_1 C_2}{C_1 + C_2}$$

$$f_{tuned} = \frac{1}{2\pi\sqrt{L_T C}}$$

$$f_{detuned} = \frac{1}{2\pi\sqrt{L_1 C}}$$

$$L_T = L_1 + L_2 + 2L_m$$

Bandwidth requirement and limit on circuit Q for MCRF355

Since the MCRF355 operates with a data rate of 70 kHz, the reader antenna circuit needs a bandwidth of at least twice of the data rate. Therefore, it needs:

EQUATION 32:

$$B_{minimum} = 140 \text{ kHz}$$

Assuming the circuit is turned at 13.56 MHz, the maximum attainable Q is obtained from Equations 31 and 32:

EQUATION 33:

$$Q_{max} = \frac{f_o}{B} = 96.8$$

In a practical LC resonant circuit, the range of Q for 13.56 MHz band is about 40. However, the Q can be significantly increased with a ferrite core inductor. The system designer must consider the above limits for optimum operation.

RESONANT CIRCUITS

Once the frequency and the inductance of the coil are determined, the resonant capacitance can be calculated from:

EQUATION 34:

$$C = \frac{1}{L(2\pi f_o)^2}$$

In practical applications, parasitic (distributed) capacitance is present between turns. The parasitic capacitance in a typical tag antenna coil is a few (pF). This parasitic capacitance increases with operating frequency of the device.

There are two different resonant circuits: parallel and series. The parallel resonant circuit has maximum impedance at the resonance frequency. It has a minimum current and maximum voltage at the resonance frequency. Although the current in the circuit is minimum at the resonant frequency, there are a circulation current that is proportional to Q of the circuit. The parallel resonant circuit is used in both the tag and the high-power reader antenna circuit.

On the other hand, the series resonant circuit has a minimum impedance at the resonance frequency. As a result, maximum current is available in the circuit. Because of its simplicity and the availability of the high current into the antenna element, the series resonant circuit is often used for a simple proximity reader.

Parallel Resonant Circuit

Figure 14 shows a simple parallel resonant circuit. The total impedance of the circuit is given by:

EQUATION 35:

$$Z(j\omega) = \frac{j\omega L}{(1-\omega^2 LC) + j\frac{\omega L}{R}} \quad (\Omega)$$

where ω is an angular frequency given as $\omega = 2\pi f$.

The maximum impedance occurs when the denominator in the above equation is minimized. This condition occurs when:

EQUATION 36:

$$\omega^2 LC = 1$$

This is called a resonance condition, and the resonance frequency is given by:

EQUATION 37:

$$f_0 = \frac{1}{2\pi\sqrt{LC}}$$

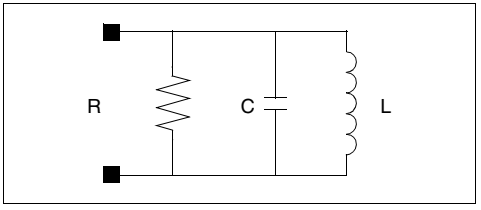
By applying Equation 36 into Equation 35, the impedance at the resonance frequency becomes:

EQUATION 38:

$$Z = R$$

where R is the load resistance.

FIGURE 14: PARALLEL RESONANT CIRCUIT



The R and C in the parallel resonant circuit determine the bandwidth, B, of the circuit.

EQUATION 39:

$$B = \frac{1}{2\pi RC} \qquad (Hz)$$

The quality factor, Q, is defined by various ways such as

EQUATION 40:

$$Q = \frac{\text{Energy Stored in the System per One Cycle}}{\text{Energy Dissipated in the System per One Cycle}}$$

$$= \frac{reactance}{resistance}$$

$$= \frac{\omega L}{r} \qquad \text{For inductance}$$

$$= \frac{1}{\omega cr} \qquad \text{For capacitance}$$

$$= \frac{f_0}{B}$$

where:

- ω = $2\pi f$ = angular frequency
- f_o = resonant frequency
- B = bandwidth
- r = ohmic losses

By applying Equation 37 and Equation 39 into Equation 40, the Q in the parallel resonant circuit is:

EQUATION 41:

$$Q = R\sqrt{\frac{C}{L}}$$

The Q in a parallel resonant circuit is proportional to the load resistance R and also to the ratio of capacitance and inductance in the circuit.

When this parallel resonant circuit is used for the tag antenna circuit, the voltage drop across the circuit can be obtained by combining Equations 8 and 41:

EQUATION 42: .

$$V_o = 2\pi f_o NQSB_o \cos\alpha$$

$$= 2\pi f_0 N\left(R\sqrt{\frac{C}{L}}\right)SB_0 \cos\alpha$$

The above equation indicates that the induced voltage in the tag coil is inversely proportional to the square root of the coil inductance, but proportional to the number of turns and surface area of the coil.

Series Resonant Circuit

A simple series resonant circuit is shown in Figure 15. The expression for the impedance of the circuit is:

EQUATION 43:

$$Z(j\omega) = r + j(X_L - X_C) \qquad (\Omega)$$

where:

- r = a dc ohmic resistance of coil and capacitor
- X_L *and* X_C = the reactance of the coil and capacitor, respectively, such that:

EQUATION 44:

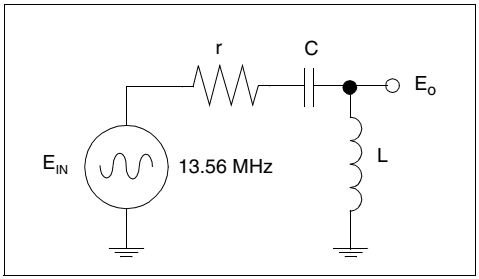
$$X_L = 2\pi f_o L \qquad (\Omega)$$

EQUATION 45:

$$X_c = \frac{1}{2\pi f_o C} \qquad (\Omega)$$

The impedance in Equation 43 becomes minimized when the reactance component cancelled out each other such that $X_L = X_C$. This is called a resonance condition. The resonance frequency is same as the parallel resonant frequency given in Equation 37.

FIGURE 15: SERIES RESONANCE CIRCUIT



The half power frequency bandwidth is determined by r and L, and given by:

EQUATION 46:

$$B = \frac{r}{2\pi L} \qquad (Hz)$$

The quality factor, Q, in the series resonant circuit is given by:

$$Q = \frac{f_0}{B} = \frac{\omega L}{r} = \frac{1}{r\omega C}$$

The series circuit forms a voltage divider, the voltage drops in the coil is given by:

EQUATION 47:

$$V_o = \frac{jX_L}{r + jX_L - jX_c} V_{in}$$

When the circuit is tuned to a resonant frequency such as $X_L = X_C$, the voltage across the coil becomes:

EQUATION 48:

$$V_o = \frac{jX_L}{r} V_{in}$$

$$= jQV_{in}$$

The above equation indicates that the coil voltage is a product of input voltage and Q of the circuit. For example, a circuit with Q of 40 can have a coil voltage that is 40 times higher than input signal. This is because all energy in the input signal spectrum becomes squeezed into a single frequency band.

EXAMPLE 6: CIRCUIT PARAMETERS

If the DC ohmic resistance r is 5 Ω, then the L and C values for 13.56 MHz resonant circuit with Q = 40 are:

EQUATION 49:

$$X_L = Qr_s = 200\Omega$$

$$L = \frac{X_L}{2\pi f} = \frac{200}{2\pi(13.56MHz)} = 2.347 \qquad (\mu H)$$

$$C = \frac{1}{2\pi f X_L} = \frac{1}{2\pi(13.56\ \text{MHz})(200)} = 58.7 \ \text{(pF)}$$

6

RFID

TUNING METHOD

The circuit must be tuned to the resonance frequency for a maximum performance (read range) of the device. Two examples of tuning the circuit are as follows:

- **Voltage Measurement Method:**
  a) Set up a voltage signal source at the resonance frequency.
  b) Connect a voltage signal source across the resonant circuit.
  c) Connect an Oscilloscope across the resonant circuit.
  d) Tune the capacitor or the coil while observing the signal amplitude on the Oscilloscope.
  e) Stop the tuning at the maximum voltage.

- **S-parameter or Impedance Measurement Method using Network Analyzer:**
  a) Set up an S-Parameter Test Set (Network Analyzer) for S11 measurement, and do a calibration.
  b) Measure the S11 for the resonant circuit.
  c) Reflection impedance or reflection admittance can be measured instead of the S11.
  d) Tune the capacitor or the coil until a maximum null (S11) occurs at the resonance frequency, f_o. For the impedance measurement, the maximum peak will occur for the parallel resonant circuit, and minimum peak for the series resonant circuit.

FIGURE 16: VOLTAGE VS. FREQUENCY FOR RESONANT CIRCUIT

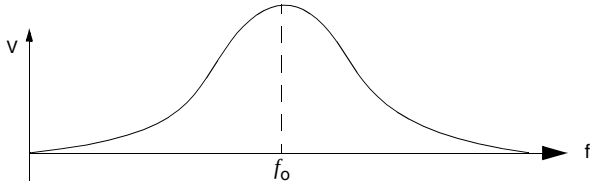
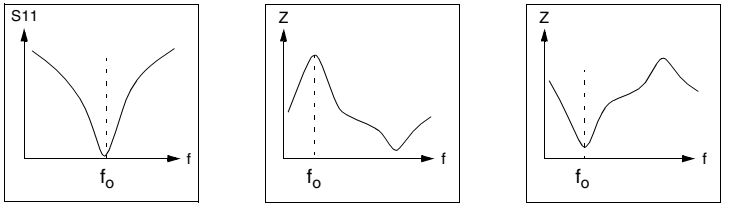


FIGURE 17: FREQUENCY RESPONSES FOR RESONANT CIRCUIT



(a) (b) (c)

Note 1: (a) S11 Response, (b) Impedance Response for a Parallel Resonant Circuit, and (c) Impedance Response for a Series Resonant Circuit.

2: In (a), the null at the resonance frequency represents a minimum input reflection at the resonance frequency. This means the circuit absorbs the signal at the frequency while other frequencies are reflected back. In (b), the impedance curve has a peak at the resonance frequency. This is because the parallel resonant circuit has a maximum impedance at the resonance frequency. (c) shows a response for the series resonant circuit. Since the series resonant circuit has a minimum impedance at the resonance frequency, a minimum peak occurs at the resonance frequency.

READ RANGE OF RFID DEVICES

Read range is defined as a maximum communication distance between the reader and tag. In general, the read range of passive RFID products varies, depending on system configuration and is affected by the following parameters:

a) Operating frequency and performance of antenna coils
b) *Q* of antenna and tuning circuit
c) Antenna orientation
d) Excitation current
e) Sensitivity of receiver
f) Coding (or modulation) and decoding (or demodulation) algorithm
g) Number of data bits and detection (interpretation) algorithm
h) Condition of operating environment (electrical noise), etc.

The read range of 13.56 MHz is relatively longer than that of 125 kHz device. This is because the antenna efficiency increases as the frequency increases. With a given operating frequency, the conditions (a – c) are related to the antenna configuration and tuning circuit. The conditions (d – e) are determined by a circuit topology of reader. The condition (f) is a communication protocol of the device, and (g) is related to a firmware software program for data detection.

Assuming the device is operating under a given condition, the read range of the device is largely affected by the performance of the antenna coil. It is always true that a longer read range is expected with the larger size of the antenna with a proper antenna design. Figures 18 and 19 show typical examples of the read range of various passive RFID devices.

FIGURE 18: READ RANGE VS. TAG SIZE FOR TYPICAL PROXIMITY APPLICATIONS*

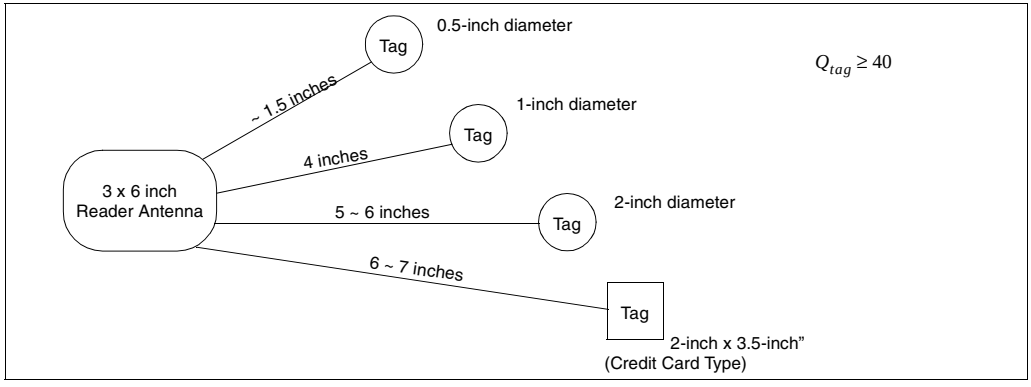
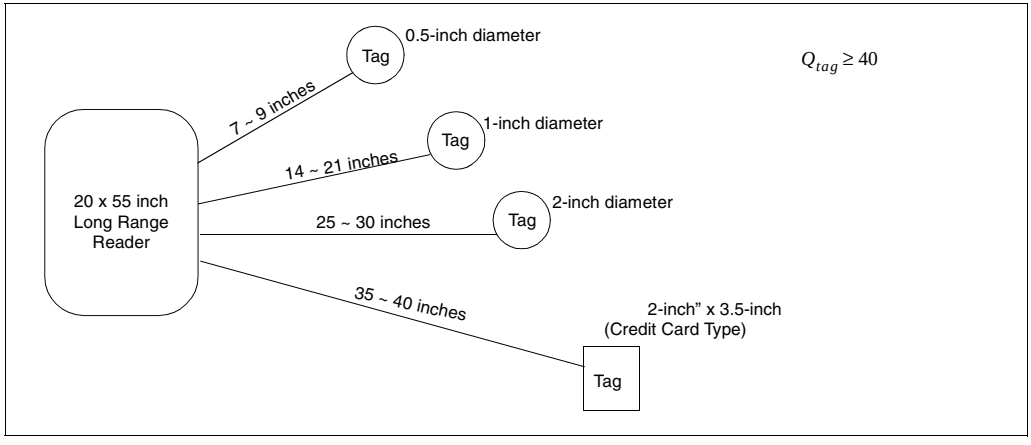


FIGURE 19: READ RANGE VS. TAG SIZE FOR TYPICAL LONG RANGE APPLICATIONS*



Note: Actual results may be shorter or longer than the range shown, depending upon factors discussed above.

REFERENCES

[1] V. G. Welsby, The Theory and Design of Inductance Coils, John Wiley and Sons, Inc., 1960.

[2] Frederick W. Grover, Inductance Calculations Working Formulas and Tables, Dover Publications, Inc., New York, NY., 1946.

[3] Keith Henry, Editor, Radio Engineering Handbook, McGraw-Hill Book Company, New York, NY., 1963.

[4] James K. Hardy, High Frequency Circuit Design, Reston Publishing Company, Inc.Reston, Virginia, 1975.

AN725

Optimizing Read-Range of the 13.56 MHz Demonstration Reader

Author: *Youbok Lee, Ph.D*
Microchip Technology Inc.

INTRODUCTION

The 13.56 MHz Anticollision Reader in the DV103003 microID™ Developer's Kit is designed to demonstrate basic operation of the MCRF355, but not to show the limits of its performance. The MCRF355 is a very advanced, carrier-independent tagging IC with the lowest power consumption and highest speed in the industry as of this writing. Designing a reader that takes advantage of the inherent performance of the MCRF355 involves two primary optimizations:

a) Increasing the speed of the digital processing by using a high-end PICmicro® microcontroller (MCU) to sample the data and calculate the checksums. This will help take advantage of the 2.2 msec data burst time and high-speed anticollision capabilities of the MCRF355.

b) Increasing the reader's carrier field volume and/or power output in order to provide power to the tag at longer distance from the reader. This application note describes one method of accomplishing this improvement.

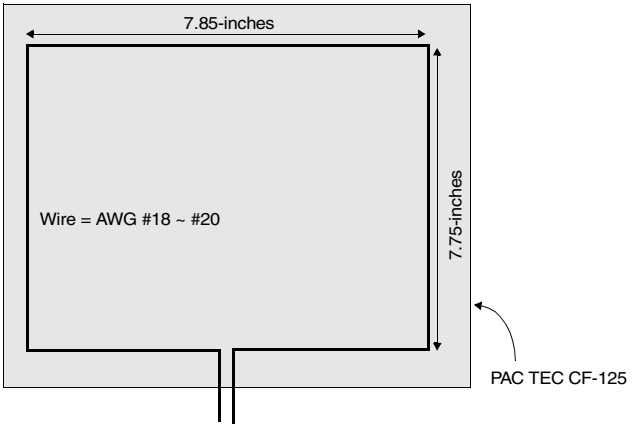
Following are the steps to achieve a read-range of 12 inches to 18 inches using a 2-inch x 2-inch sample tag based on MCRF355, properly tuned to the carrier frequency.

1. Disconnect the power cable and RS-232 cable from the reader, and remove the six screws from the back of the case.
2. Make an antenna with the following parameters:
   a) Use AWG #18 ~ #20 wire.
   b) Make one turn: a rectangular loop with 7.85-inches x 7.75-inches as shown in Figure 1. This antenna will fit in PAC TEC's CF-125 enclosure. The enclosure is available from PAC TEC or its distributors. This will result in about 800 nH ~ 1 µH of inductance.
   c) This inductance requires 172 pF ~ 138 pF of capacitance to tune the antenna circuit to 13.56 MHz.
3. Connect the new inductor (antenna) and capacitor to the demo reader board by following these steps:
   a) Disconnect the C31, C9, and C10 from the circuit board.
   b) Disconnect L3 (printed antenna) from the circuit. This can be accomplished by cutting off the metallic trace on the board.
   c) Connect the new resonant capacitance (172 pF ~ 138 pF) at C31, C9, C10.
   d) Connect the new antenna at L3. Connect one side of the antenna to the resonant capacitor and the other side to ground.
4. Tuning the antenna circuit:

   The benefit of this modification will be realized only if the antenna circuit is tuned precisely to the 13.56 MHz carrier. Here is one method for tuning the circuit:
   a) Connect an oscilloscope across the new antenna (L3).
   b) Observe the voltage while adjusting the capacitance (C31, C9, C10).
   c) Adjust the cap to **+** and **-** direction and stop at the maximum voltage.
   d) Bring the voltage to above 200 VPP.
5. Read-Range Measurement:

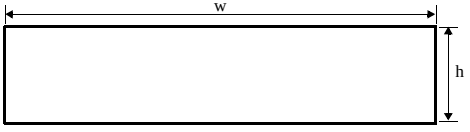
   Reconnect the power and RS-232 cables to the reader. The reader should now provide between 12 ~ 18 inches of read range. If it does not exhibit this performance, check the following:
   a) Check the antenna voltage again, bringing it to above 200 VPP.
   b) Adjust VR1 in the reader circuit; VR1 is very sensitive to voltage. Connect a 1 MΩ resistor across C17 permanently. Then, connect a Digital Volt Meter across the resistor, and adjust the VR1 between 4.7-volts to 4.87-volts while measuring the read range. Set the VR1 for maximum range.

FIGURE 1: ANTENNA GEOMETRY



1.1 Formula for Inductance Calculation

EQUATION 1: RECTANGULAR LOOP

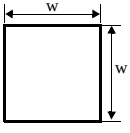


$$L_{rect}[nH] = (N^2)(10.16)\left[-2(w+h)+2\sqrt{h^2+w^2}-h\ln\left(\frac{h+\sqrt{h^2+w^2}}{w}\right)-w\ln\left(\frac{w+\sqrt{h^2+w^2}}{h}\right)+h\ln\left(\frac{2w}{a}\right)+w\ln\left(\frac{2h}{a}\right)\right]$$

where:

N = number of turns
w = width of the rectangle (inches)
h = height of the rectangle (inches)
a = wire radius (inches)

EQUATION 2: SQUARE LOOP



$$L_{square}[nH] = (N^2)(20.32)w\left[\ln\left(\frac{w}{a}\right)-0.774\right]$$

where:

N = number of turns
w = length of one side (inches)
a = wire radius (inches)

TB019

Contactless Programmer Interface Protocol

Author: *Shannon Poulin*
Microchip Technology Inc.

INTRODUCTION

The following is a description of how to interface to Microchip's contactless programmer for use with the MCRF2xx family of RF/ID products. The programmer will check for a blank, unlocked MCRF2xx tag before initiating programming. Once programming has been completed, the programmer will return a pass or fail code. The programmer communicates at 9600 baud, 8 data bits, 1 stop bit, and no parity.

Programmer Wake-up

Sending an ASCII 'W' (57hex) to the programmer on the RS-232 interface will tell the programmer to wake up and be prepared to receive commands. The programmer will reply with ASCII 'R' (52h) when it is ready.

Blank Check

Sending an ASCII 'T' (54h) will signal the programmer to read the part about to be contactlessly programmed and check to see if it is blank (all 1's) and unlocked. If the part is blank and unlocked the programmer will reply with an ASCII 'Y' (59h) to signify programming should continue. If the part is not blank or not unlocked, the programmer will reply with an ASCII 'N' (4Eh) to indicate an error. It is always necessary to perform a blank check before programming MCRF2xx devices.

Sending Data to the Programmer

If the programmer responds with an ASCII 'Y' to indicate the part is blank, the PC can begin passing the 16 bytes of required data to the programmer data buffer. The data should be passed in ASCII equivalent hex bytes and the programmer will acknowledge the receipt of each byte by echoing back what it has received. For example, to program 05 hex data into the first byte the PC would send ASCII '0' (30h), the programmer would echo 0 back. Next the programmer would send ASCII 5 (35h), and the programmer will echo back 5. All of the data must be sent in UPPERCASE ASCII equivalent only. See Figure 1 for a typical programming sequence.

Program and Verify the Device

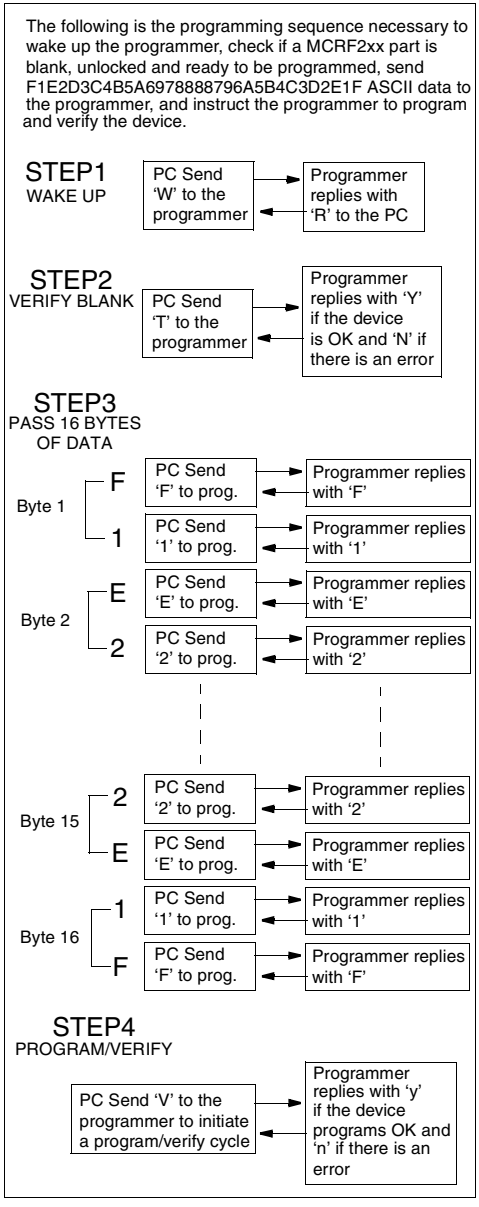
After 16 bytes of data have been received by the programmer, it is ready to begin programming the data buffer into the MCRF2xx. Sending an ASCII 'V' (56h) will tell the programmer to program the 16 bytes it has received and verify that the device has programmed properly. When the device programs properly, the programmer replies with ASCII 'y' (79h). If the programming was not successful, the programmer replies with ASCII 'n' (6Eh). A successful programming operation should take less than 2 seconds per device.

Error Conditions

If the PC does not send a byte to the programmer for more than 3 seconds, the programmer will timeout and reset. The entire programming sequence will need to be repeated, beginning with the programmer wake-up byte ASCII 'W'.

If invalid bytes are sent to the programmer during the loading of the program buffer, the programmer will return an ASCII 'Q' (51h). The entire programming sequence will need to be repeated, beginning with the programmer wake-up byte ASCII 'W'.

FIGURE 1: TYPICAL SEQUENCE



ASCII CHARACTER SET

		Most Significant Characters							
	Hex	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Least Significant Characters	0	NUL	DLE	Space	0	@	P	‘	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	Bell	ETB	’	7	G	W	g	w
	8	BS	CAN	(	8	H	X	h	x
	9	HT	EM	)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[	k	{
	C	FF	FS	,	<	L	\	l	|
	D	CR	GS	-	=	M	]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

NOTES:

TB023

Contact Programming Support

Author: *Pete Sorrells*
Microchip Technology Inc.

INTRODUCTION

The MCRF200 and MCRF250 are 125 kHz RF tags, which can be contact or contactlessly programmed. The contact programming of the device is performed by Microchip Technology, Inc. upon customer request. The customer can choose any ID code suitable to their application subject to a minimum order quantity. These devices can also be contactlessly programmed after encapsulation using the Microchip microID contactless programmer (PG103001).

DEFINITIONS

First, the customer has to define the following operation options of the MCRF200 (DS21219) and MCRF250 (DS21212):

- Bit rate — Defined as clocks per bit e.g., Fc/16, Fc/32, Fc/40, Fc/50, Fc/64, Fc/80, Fc/100, and Fc/128
- Modulation — FSK, PSK1, PSK2, ASK Direct
- Encoding — NRZ_L (Direct), Biphase_L (Manchester), Differential Biphase_S
- Code length — 32, 48, 64, 96, and 128 bits

Second, the ID codes and series numbers must be supplied by the customer or an algorithm can be specified by the customer. This section describes only the case in which actual serial numbers are supplied. The customer must supply the ID codes and series numbers on floppy disk or via email. The codes should conform to the SQTP format below:

FILE SPECIFICATION

SQTP codes supplied to Microchip must comply with the following format:

The ID code file is a plain ASCII text file from floppy disk or email (no headers).

The code files should be compressed. Please make self-extracting files.

The code files are used in alphabetical order of their file names (including letters and numbers).

Used (i.e., programmed) code files are discarded by Microchip after use.

Each line of the code file must contain one ID code for one IC.

The code is in hexadecimal format.

The code line is exactly as long as the selected code length (e.g., code length = 64, ID code = 16 hex characters = 64-bit number).

Each line must end with a carriage return.

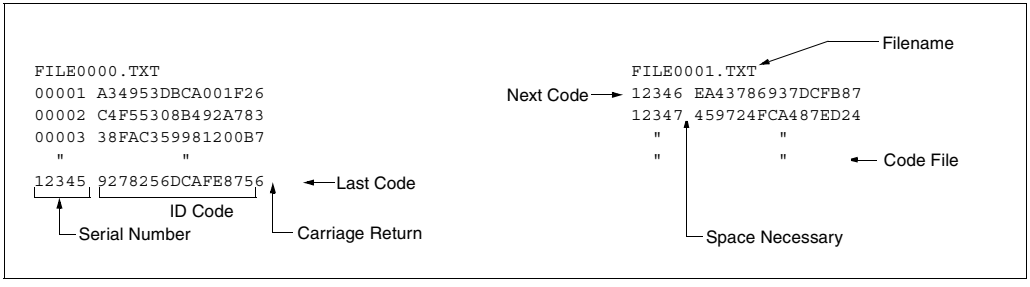
Each hexadecimal ID code must be preceded by a decimal series number.

Series number and ID code must be separated by a space.

The series number must be unique and ascending to avoid double programming.

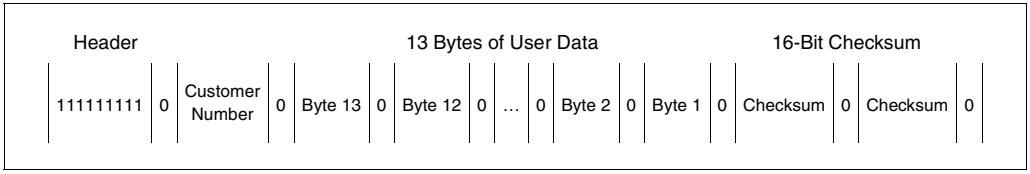
The series numbers of two consecutive files must also count up for proper linking.

FIGURE 1: EXAMPLE OF TWO CODE FILES, CODE LENGTH = 64 BITS



NOTES:

Microchip Development Kit Sample Format



9 bit header	
8 bit customer number	
104 bits (13 x 8) of user data	
17 bits of zeros between each byte, header, and checksum	
16 bits of checksum	
Total:	154 bits

Notes:

- Users can program all 154 bits of the MCRF355/360. The array can be programmed in any custom format and with any combination of bits.
- The format presented here is used for Microchip microID™ Development System (DV103003) and can be ordered as production material with a unique customer number.
- See TB032 for information on ordering custom programmed production material.
- The Microchip Development System (DV103003) uses nine 1's (111111111) as header.
- The preprogrammed tag samples in the development kit have hex 11(= 0001 0001) as the customer number.
- For the development system, users can program the customer number (1 byte) plus the 13 bytes of user data, or they can deselect the "Microchip Format" option in the MicroID™ RFLAB and program all 154 bits in any format.
- When users program the samples using the MicroID™ RFLAB, the RFLAB calculates the checksum (2 bytes) automatically by adding up all 14 bytes (customer number + 13 bytes of user data), and put into the checksum field in the device memory. See Example 1 for details.
- When the programmed tag is energized by the reader field, the tag outputs all 154 bits of data.
- When the demo reader detects data from the tag, it reports the 14 bytes of the data (customer number plus 13 bytes of user data) to the host computer if the header and checksum are correct. The reader does not send the header and checksum to the host computer.
- The "MicroID™ RFLab" or a simple terminal program such as "terminal.exe" can be used to read the reader's output (28 hex digits) on the host computer.
- When the demo reader is used in the terminal mode ("terminal.exe), the tag's data appear after the first two dummy ASCII characters (GG). See Example 2 for details.

EXAMPLE 1: CHECKSUM

Checksum (xxxxxxxx xxxxxxxx) = Byte 1 + Byte 2 + ...+ Byte 13 + Customer Number (1 byte)

EXAMPLE 2: READER'S OUTPUT IN TERMINAL MODE ("TERMINAL.EXE")

The demo reader outputs GG+28 hex digits, i.e., GG 12345678901234567890ABCDEFGF.

The first two ASCII characters (GG) are dummy characters.

The tag's data are the next 28 hex digits (112 bits) after the first two ASCII characters (GG).

NOTES:

TB032

MCRF355/360 Factory Programming Support (SQTP^{SM})

Author: *Shannon Poulin*
Microchip Technology Inc.

INTRODUCTION

The MCRF360 and MCRF360 are 13.56 MHz RF tags which can be contact programmed. The contact programming of the device can be performed by the user or factory-programmed by Microchip Technology, Inc. upon customer request. All 154 bits of data may be programmed in any format or pattern defined by the customer.

For factory programming, ID codes and series numbers must be supplied by the customer or an algorithm may be specified by the customer. This technical brief describes only the case in which identification codes (ID) and series numbers are supplied. The customer may supply the ID codes and series numbers on floppy disk or via email. The codes must conform to the Serialized Quick Turn Programming^{SM} (SQTP^{SM}) format below:

FILE SPECIFICATION

SQTP codes supplied to Microchip must comply with the following format:

The ID code file is a plain ASCII text file from floppy disk or email (no headers).

If code files are compressed, they should be self-extracting files.

The code files are used in alphabetical order of their file names (including letters and numbers).

Used (i.e., programmed) code files are discarded by Microchip after use.

Each line of the code file must contain one ID code for one IC.

The code is in hexadecimal format.

The code line is exactly 154 bits (39 hex characters, where the last 2 bits of the last character are don't cares).

Each line must end with a carriage return.

Each hexadecimal ID code must be preceded by a decimal series number.

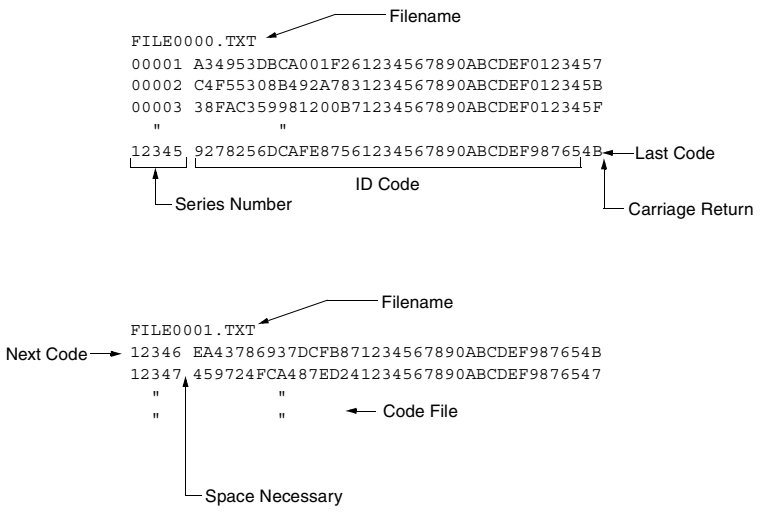
Series number and ID code must be separated by a space.

The series number must be unique and ascending to avoid double programming.

The series numbers of two consecutive files must also count up for proper linking.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology inc.

FIGURE 1: EXAMPLE OF TWO SEQUENTIAL CODE FILES



SECTION 7
REFERENCE DESIGNS

PICREF-1

Uninterruptible Power Supply Reference Design

INTRODUCTION

At times, power from a wall socket is neither clean nor uninterruptible. Many abnormalities such as blackouts, brownouts, spikes, surges, and noise can occur. Under the best conditions, power interruptions can be an inconvenience. At their worst, they can cause loss of data in computer systems or damage to electronic equipment.

It is the function of an Uninterruptible Power Supply (UPS) to act as a buffer and provide clean, reliable power to vulnerable electronic equipment. The basic concept of a UPS is to store energy during normal operation (through battery charging) and release energy (through DC to AC conversion) during a power failure.

UPS systems are traditionally designed using analog components. Today these systems can integrate a microcontroller with AC sine wave generation, offering the many benefits listed below.

PIC17C43 Microcontroller Benefits

- High Quality Sine Wave - High throughput allows for high quality output
- Flexibility - core control features and operations can be changed with software modifications only
- Transportability of Design
- Variable Loop Response
- Digital Filtering
- Parts and Complexity Reduction
- Peripheral Integration
- Ease of Interfacing
- Testability
- Time to Market

PICREF-1 OVERVIEW

The Microchip Technology PICREF-1 UPS Reference Design offers a ready-made uninterruptible power supply solution with the flexibility of a microcontroller.

The PIC17C43 microcontroller handles all the control of the UPS system. The PIC17C43 is unique because it provides a high performance and low cost solution not found in other microcontrollers.

The PIC17C43 PWM controls an inverter whose output, when filtered, results in a sinusoidal AC output waveform. Fault signaling can be initiated internal or external to the PIC17C43 depending on the type of fault. A fault will disable the entire inverter. The output voltage and current will be monitored by the PIC17C43 to make adjustments “real-time” to correct for DC offset and load changes.

The PIC17C43 controls all module synchronization as well as inverter control and feedback. The PIC17C43 uses zero crossing for synchronization of input voltage/phase to output voltage/phase. All internal module synchronization is handled by the PIC17C43.

The control algorithms and software are written in C for maintainability and transportability.

PICREF-1 Key Features

- True UPS Topology
- True Sinusoidal Output
- Point-to-Point Output Correction
- 1400 VA Rating
- 120/240 V Input

For complete documentation of DS30450C, PICREF1, *Uninterruptible Power Supply Reference Design*, please visit our worldwide website @www.microchip.com

NOTES:

ACKNOWLEDGMENTS

Project Lead Engineer:
Robert Schreiber, Microchip Technology

Reference Design Documentation:
Beth McLoughlin, Microchip Technology

System and Code Development:
Airborne Power (Consultants)
Guy Gazia (guyg@airbornepower.com),
David Karipides (davek@airbornepower.com),
Terry Allinder

PICREF-2

Intelligent Battery Charger Reference Design

INTRODUCTION

Typically, simple battery chargers do not provide the intelligence to charge different battery technologies or batteries with the same technology but different voltages and capacities. At best, this may leave the battery improperly charged. At worst, it can pose a serious safety hazard. A microcontroller can provide the intelligence to overcome these problems.

In addition to intelligent control, the microcontroller can provide a low-cost, flexible solution for charging batteries. Complete battery charging applications may be developed quickly using a microcontroller. Add to this the serial communication capability of the microcontroller, real-time data logging and monitoring is possible.

Simple battery chargers use all analog components to accomplish their function. However, by using a microcontroller, a battery charger can be made intelligent.

Microcontroller Benefits

- Flexibility to handle different technologies, voltages and capacities.
- Variable Voltage Generation Control
- Charge/Discharge Multiple Battery Packs
- “Windowed” A/D for High Resolution

PICREF-2 OVERVIEW

The Microchip Technology PICREF-2 Intelligent Battery Charger (IBC) Reference Design offers a ready-made battery charger solution. This Reference Design is targeted to battery charger applications such as camcorders, portable audio equipment, portable phones, and portable power tools.

With the PICREF-2 Reference Design, the user will be able to simply pick their complete battery charging system by completing the steps listed:

1. Pick the required battery management features from the modular source code provided.
2. Pick the critical battery pack parameters and modify the global constants to those specifications.

The hardware design contains the necessary circuitry to support charging and discharging algorithms, charge termination methods, and RS-232 communications.

The modular source code is written in C and consists of the charge termination algorithms, discharge algorithm, interdevice communications, and RS-232 communications modules.

The PC based software provides a means for requesting and displaying battery status information.

PICREF-2 Key Features

- Compatibility Across Battery Technologies
- Low Cost
- Flexible Development Environment
- Fast Charge Rate
- High Charge Current Capability
- High Discharge Current Capability for Conditioning
- Real-Time Debug
- Data Logging
- User Selectable Embedded Charge Termination Algorithms

For complete documentation of DS30145C, *PICREF-2, Intelligent Battery Charger Reference Design*, please visit our worldwide website @www.microchip.com

NOTES:

ACKNOWLEDGMENTS

Project Lead Engineer:
Robert Schreiber,
Microchip Technology, Inc.

Reference Design Documentation:
Beth McLoughlin,
Microchip Technology, Inc.

System and Code Development:
TriSys Inc.,Consultants

TRADEMARKS

PICREF-3

Watt-Hour Meter Reference Design

INTRODUCTION

The PICREF-3 Watt-Hour Meter (WHM) Reference Design shows the use of a mixed signal microcontroller in an AC power measurement application.

The traditional sensor signal processing chain consisting of sensor, signal conditioning electronics, A/D converter and microcontroller is abbreviated by the use of the mixed signal microcontroller with its on-board A/D converter.

The mixed signal microcontroller used is the Microchip PIC16C924. This microcontroller has five A/D channels, two of which are used to digitize voltage and current signals. The microcontroller features of pulse width modulation (PWM) and direct liquid crystal display (LCD) drive are utilized to further reduce cost and parts count.

The PWM output feature is used with a single pole RC filter to provide a comparator reference with 10 bits of resolution.

The direct LCD drive is used to drive an 8-digit, 7-segment LCD.

MICROCONTOLLER BENEFITS

The use of a PIC16C924 microcontroller in a power meter offers the following advantages:

- Real-Time Electrical Measurement and Power/Energy Calculations
- Direct LCD Drive
  - Present Time
  - Total Watt-Hours (Whr)
  - Maximum or Cumulative Demand
- Customization
- Quick Time-to-Market

PICREF-3 OVERVIEW

The WHM Reference Design provides a cost effective circuit capable of monitoring and displaying power and energy consumption on worldwide power mains in the 90V to 264V range.

The PIC16C924 microcontroller shows that the real-time events of sampling voltage and current waveforms can be interleaved with power and energy calculations. All measurements and calculations are performed once per second.

The current waveforms measured are linear for resistive and inductive loads and non-linear for switching power supplies. The current waveform is sampled during the positive current cycle with waveform symmetry assumed between positive and negative cycles (valid for the measured waveforms). A hardware method for full cycle current measurements and firmware methods for complex current waveform shapes are provided in the *Design Modifications* section.

PICREF-3 KEY FEATURES

- Accepts polarized and unpolarized worldwide power mains.
- Measures and displays AC Voltage (90V to 264V), Load Current and Power Factor.
- Measures power line frequency (47 Hz to 63 Hz).
- Calculates Watts, Watt-Hrs and cumulative Watt-Hrs and displays these values, as well as frequency and time.
- True RMS measurements.
- Firmware control of triac load switch on/off state.
- Real time clock during power-saving sleep mode.
- Hibernate mode to save on battery life during storage.
- Battery back-up for microcontroller.

For complete documentation of DS30452A, *PICREF-1, Watt-Hour Meter Reference Design*, please visit our worldwide website @www.microchip.com

NOTES:

ACKNOWLEDGMENTS

Hardware Design and Firmware Development:
Dennis E. Coleman, Sr. Applications Engineer,
Microchip Technology, Inc.
dennis.coleman@microchip.com

Documentation:
Beth McLoughlin, Applications Engineer,
Microchip Technology, Inc.

PICREF-4

PICDIM Lamp Dimmer for the PIC12C508

INTRODUCTION

The PIC12CXXX family of devices adds a new twist to the 8-bit microcontroller market by introducing for the first time fully functional microcontrollers in an eight pin package. These parts are not stripped down versions of their larger brethren, they add features in a package smaller than available ever before for microcontrollers. Using the familiar 12-bit opcode width of the PIC16C5X family with the same TMR0 module, Device Reset Timer, and WatchDog Timer (WDT), the PIC12C5XX family adds an internal 4MHz oscillator main clock, serial programming, wake-up on change, user selectable weak pullups, and multiplexing of the MCLR, T0CKI, OSC1, and OSC2 pins.

This combination of familiar and new features in a compact package gives the designer unprecedented flexibility to produce designs which are much cheaper and smaller than ever before possible, and allows the replacement of even mundane devices like timers and discrete components economically.

This reference note describes an application where the use of a microcontroller was not previously economically feasible for any but the highest end products: lamp dimming.

For complete documentation of DS40171A, *PICREF-4, PICDIM Lamp Dimmer for the PIC12C508 Reference Design*, please visit our worldwide website @www.microchip.com

NOTES:

ACKNOWLEDGMENTS

Project Lead Engineer:
Scott Fink

System and Code Development:
Scott Fink

13.56 MHz Reader Reference Design

1.0 INTRODUCTION

This chapter provides a reference guide for the 13.56 MHz reader designer. The schematic included in this chapter is for the 13.56 MHz Reference Reader included in the DV103003 microID™ Developer's Kit. The circuit is designed for short read-range applications. The basic design can be modified for long-range or other applications with MCRF355/360 devices. An electronic copy of the PICmicro® microcontroller source code is available upon request.

2.0 READER CIRCUITS

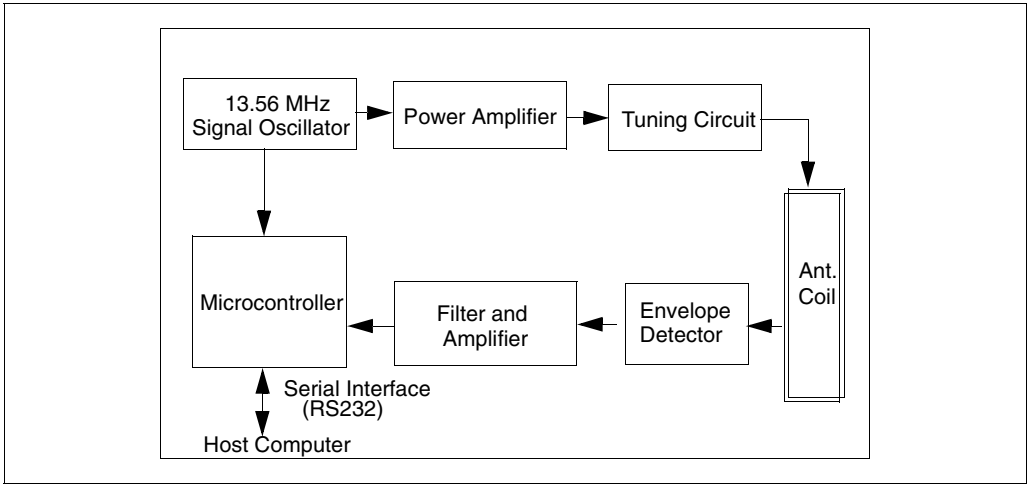
The RFID reader consists of transmitting and receiving sections. It transmits a carrier signal (13.56 MHz), receives the backscattered signal from the tag, and performs data processing. The reader also communicates with an external host computer. A basic block diagram of a typical RFID reader is shown in Figure 2-1.

The transmitting section contains a 13.56 MHz signal oscillator (74HC04), power amplifier (Q2), and RF tuning circuits. The tuning circuit matches impedance between the antenna coil circuit and the power driver at 13.56 MHz. The radiating signal strength from the antenna must comply with government regulations. For best performance, the antenna coil circuit must be tuned to the same frequency of the tag. The design for antenna circuits is given in Application Note AN710 (DS00710).

The receiving section contains an envelope detector (D6), hi-pass filters, and amplifiers (U2 and U3). When the tag is energized, it transmits 154 bits of data that is encoded in Biphase-L (Manchester). In the Manchester encoding, data '1' is represented by a logic high-to-low level change at midclock, and data '0' is represented by a low-to-high level change at midclock. There is always a level change at middle of every bit clock.

For complete documentation of DS21311A, microID™ 13.56 MHz Design Guide, please visit our worldwide website @www.microchip.com.

FIGURE 2-1: FUNCTIONAL BLOCK DIAGRAM OF TYPICAL RFID READER



NOTES:

FSK Reader Reference Design

1.0 INTRODUCTION

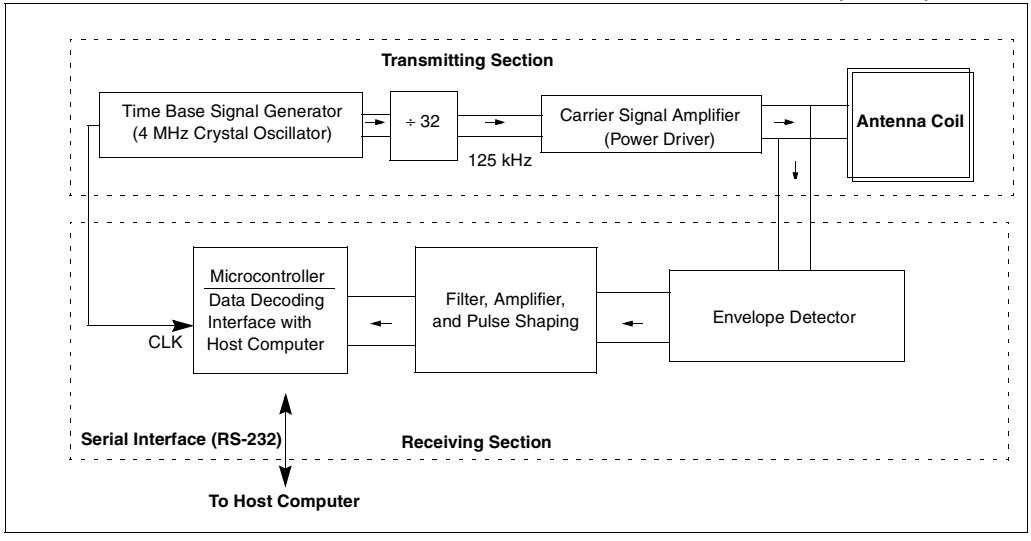
This application note is written as a reference guide for FSK reader designers. Microchip Technology Inc. provides basic reader electronics circuitry for the MCRF200 customers as a part of this design guide. The circuit is designed for a read range of 3 ~ 5 inches with an access control card. The microID FSK Reader (demo unit), which is built based on the FSK reference design, is available in the microID Designers Kit (DV103001). The circuit can be modified for longer read range or other applications with the MCRF200. An electronic copy of the FSK microID PICmicro® source code is available upon request.

2.0 READER CIRCUITS

The RFID reader consists of transmitting and receiving sections. It transmits a carrier signal, receives the backscattering signal, and performs data processing. The reader also communicates with an external host computer. A basic block diagram of the typical RFID reader is shown in Figure 2-1.

For complete documentation of DS51137B, microID™ 125 kHz Design Guide, please visit our worldwide website @www.microchip.com.

FIGURE 2-1: BLOCK DIAGRAM OF TYPICAL RFID READER FOR FSK SIGNAL (125 kHz)



PICmicro is a registered trademark of Microchip Technology Inc.

NOTES:

PSK Reader Reference Design

1.0 INTRODUCTION

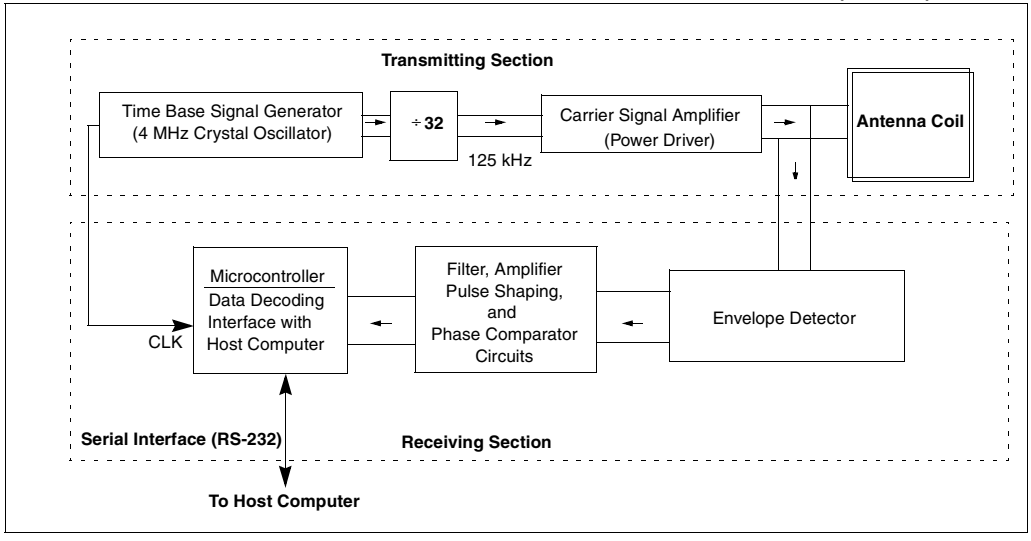
This application note is written as a reference guide for PSK reader designers. Microchip Technology Inc. provides basic reader schematic for the MCRF200 customers as a part of this design guide. The circuit is designed for a read range of 3 ~ 5 inches with an access control card. The microID PSK Reader (demo unit), which is built based on the PSK reference design, is available in the microID Designers Kit (DV103001). The circuit can be modified for longer read range or other applications with the MCRF200. An electronic copy of the PSK microID PICmicro® source code is available upon request.

2.0 READER CIRCUITS

The RFID reader consists of transmitting and receiving sections. It transmits a carrier signal, receives the backscattering signal, and performs data processing. The reader also communicates with an external host computer. A basic block diagram of the typical RFID reader is shown in Figure 2-1.

For complete documentation of DS51138B, microID™ 125 kHz Design Guide, please visit our worldwide website @www.microchip.com.

FIGURE 2-1: BLOCK DIAGRAM OF TYPICAL RFID READER FOR PSK SIGNAL (125 kHz)



PICmicro is a registered trademark of Microchip Technology Inc.

NOTES:

ASK Reader Reference Design

1.0 INTRODUCTION

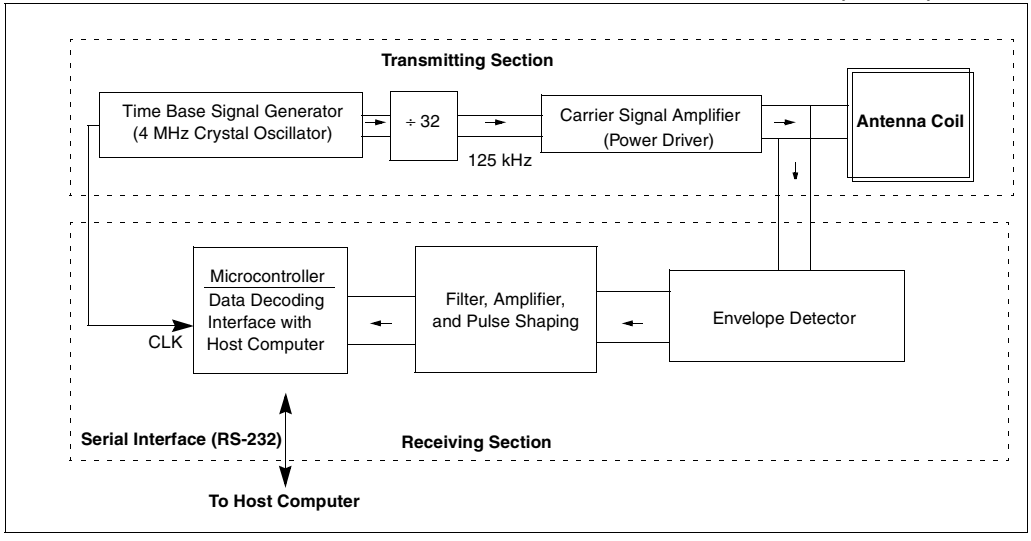
This application note is written as a reference guide for ASK reader designers. Microchip Technology Inc. provides basic reader electronics circuitry for the MCRF200 customers as a part of this design guide. The circuit is designed for a read range of 3 ~ 5 inches with an access control card. The microID ASK Reader (demo unit), which is built based on the ASK reference design, is available in the microID Designers Kit (DV103001). The circuit can be modified for longer read range or other applications with the MCRF200. An electronic copy of the ASK microID PICmicro® source code is available upon request.

2.0 READER CIRCUITS

The RFID reader consists of transmitting and receiving sections. It transmits a carrier signal, receives the backscattering signal, and performs data processing. The reader also communicates with an external host computer. A basic block diagram of the typical ASK RFID reader is shown in Figure 2-1.

For complete documentation of DS51166C, microID™ 125 kHz Design Guide, please visit our worldwide website @www.microchip.com.

FIGURE 2-1: BLOCK DIAGRAM OF TYPICAL RFID READER FOR ASK SIGNAL (125 kHz)



PICmicro is a registered trademark of Microchip Technology Inc.

NOTES:

FSK Anticollision Reader Reference Design

1.0 INTRODUCTION

When more than one tag is in the same RF field of a reader, each tag will transmit data at the same time. This results in data collision at the receiving end of the reader. No correct decision can be made based on this data. The reader must receive data from a tag at a time for correct data processing.

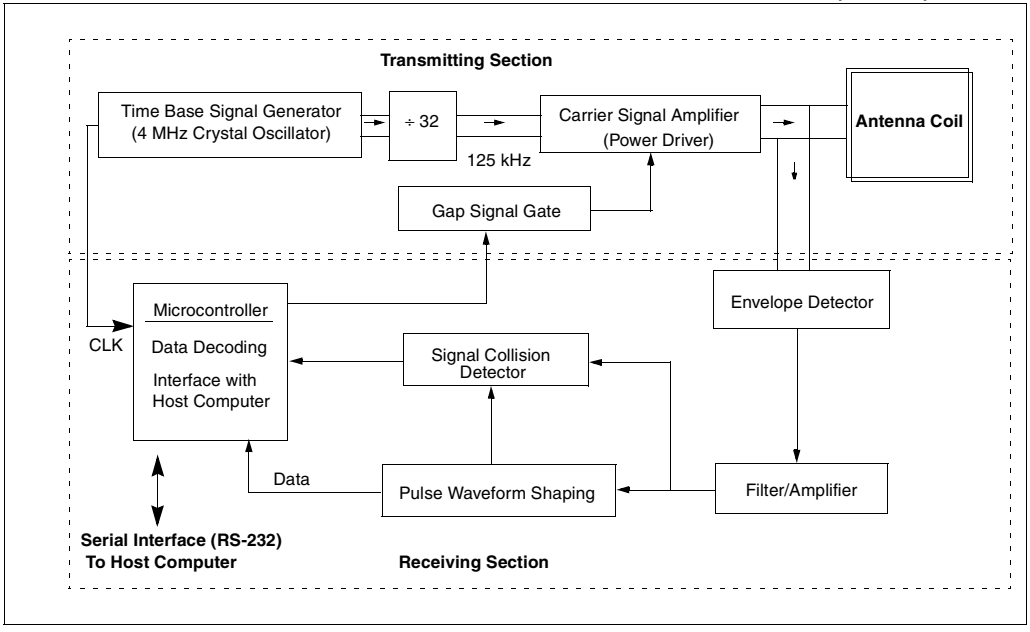
The anticollision device (MCRF250) is designed to send FSK data to reader without data collision, and it must be read by an anticollision reader. This type of device can be effectively used in inventory and asset control applications where multiple tags are read in the same RF field. The anticollision algorithm of the device is explained in the *MCRF250 Data Sheet*, page 15.

This application note is written as a reference guide for anticollision reader designers. The anticollision reader is designed to provide correct signals to the anticollision device (MCRF250) to perform an anticollision action during operation.

Microchip Technology Inc. provides basic anticollision FSK reader electronic circuitry for the MCRF250 customers as a part of this design guide. The microID Anticollision Reader (demo unit), that can read 10 tags or more in the same RF field, is available in the microID Developers Kit (DV103002). An electronic copy of the microID PICmicro® source code is also available upon request.

For complete documentation of DS51167B, microID™ 125 kHz Design Guide, please visit our worldwide website @www.microchip.com.

FIGURE 1-1: BLOCK DIAGRAM OF TYPICAL RFID READER FOR FSK SIGNAL (125 kHz)



7 Reference Designs

PICmicro is a registered trademark of Microchip Technology Inc.

NOTES:

SECTION 8
DEVELOPMENT SYSTEMS

Development Tools Selection Chart

MPLAB™-ICE Cross Reference Parts List

Model Name/ Part Number	Lead Count/ Package Type	Hardware Tools			
		Emulator Pod	Processor Module	Device Adapter	Transition Socket
PIC12C508/ PIC12C508A	8P, 8JW 8SM	ICE2000 ICE2000	PCM16XA0 PCM16XA0	DVA12XP080 DVA12XP080	— XLT08SO
PIC12C509/ PIC12C509A	8P, 8JW 8SM	ICE2000 ICE2000	PCM16XA0 PCM16XA0	DVA12XP080 DVA12XP080	— XLT08SO
PIC12CE518	8P, 8JW 8SM, 8SN	ICE2000 ICE2000	PCM16XA0 PCM16XA0	DVA12XP080 DVA12XP080	— XLT08SO
PIC12CE519	8P, 8JW 8SM, 8SN	ICE2000 ICE2000	PCM16XA0 PCM16XA0	DVA12XP080 DVA12XP080	— XLT08SO
PIC12C671	8P, 8JW 8SM	ICE2000 ICE2000	PCM12XA0 PCM12XA0	DVA12XP081 DVA12XP081	— XLT08SO
PIC12C672	8P, 8JW 8SM	ICE2000 ICE2000	PCM12XA0 PCM12XA0	DVA12XP081 DVA12XP081	— XLT08SO
PIC12CE673	8P, 8JW	ICE2000	PCM12XA0	DVA12XP081	—
PIC12CE674	8P, 8JW	ICE2000	PCM12XA0	DVA12XP081	—
PIC14C000	28SP, 28JW 28SO 28SS	ICE2000 ICE2000 ICE2000	PCM14XA0 PCM14XA0 PCM14XA0	DVA14XP280 DVA14XP280 DVA14XP280	— XLT28SO XLT28SS
PIC16C505	14P, 14JW 14SL	ICE2000 ICE2000	PCM16XA0 PCM16XA0	DVA16XP140 DVA16XP140	— XLT14SO
PIC16C52	18P 18SO	ICE2000 ICE2000	PCM16XA0 PCM16XA0	DVA16XP180 DVA16XP180	— XLT18SO
PIC16C54/ PIC16C54A	18P, 18JW 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XA0 PCM16XA0 PCM16XA0	DVA16XP181 DVA16XP181 DVA16XP181	— XLT18SO XLT20SS
PIC16HV540(*)	18P, 18JW, 18SO, 20SS	ICE2000 ICE2000 ICE2000	PCM16XM0 PCM16XM0 PCM16XM0	DVA16XP181 DVA16XP181 DVA16XP181	— XLT18SO XLT20SS
PIC16C55/ PIC16C55A	28P, 28JW 28SP 28SO 28SS	ICE2000 ICE2000 ICE2000 ICE2000	PCM16XA0 PCM16XA0 PCM16XA0 PCM16XA0	DVA16XP280 DVA16XP280 DVA16XP280 DVA16XP280	XLT28XP — XLT28SO XLT28SS2
PIC16C554	18P, 18JW 18SO 18SS	ICE2000 ICE2000 ICE2000	PCM16XC0 PCM16XC0 PCM16XC0	DVA16XP180 DVA16XP180 DVA16XP180	— XLT18SO XLT20SS
PIC16C558	18P, 18JW 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XC0 PCM16XC0 PCM16XC0	DVA16XP180 DVA16XP180 DVA16XP180	— XLT18SO XLT20SS
PIC16C56/ PIC16C56A	18P, 18JW 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XA0 PCM16XA0 PCM16XA0	DVA16XP180 DVA16XP180 DVA16XP180	— XLT18SO XLT20SS
PIC16C57/ PIC16C57C	28P, 28JW 28SP 28SO 28SS	ICE2000 ICE2000 ICE2000 ICE2000	PCM16XA0 PCM16XA0 PCM16XA0 PCM16XA0	DVA16XP280 DVA16XP280 DVA16XP280 DVA16XP280	XLT28XP — XLT28SO XLT28SS2

* Contact Microchip Technology Inc. for availability.

MPLAB™-ICE Cross Reference Parts List (Continued)

Model Name/ Part Number	Lead Count/ Package Type	Hardware Tools			
		Emulator Pod	Processor Module	Device Adapter	Transition Socket
PIC16C58A/ PIC16C58B	18P, 18JW 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XA0 PCM16XA0 PCM16XA0	DVA16XP180 DVA16XP180 DVA16XP180	— XLT18SO XLT20SS
PIC16C62A	28SP, 28JW 28SO 28SS	ICE2000 ICE2000 ICE2000	PCM16XB1 PCM16XB1 PCM16XB1	DVA16XP281 DVA16XP281 DVA16XP281	— XLT28SO XLT28SS
PIC16C62B	28SP, 28JW 28SO 28SS	ICE2000 ICE2000 ICE2000	PCM16XE1 PCM16XE1 PCM16XE1	DVA16XP281 DVA16XP281 DVA16XP281	— XLT28SO XLT28SS
PIC16C620/ PIC16C620A	18P, 18JW 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XC0 PCM16XC0 PCM16XC0	DVA16XP180 DVA16XP180 DVA16XP180	— XLT18SO XLT20SS
PIC16C621/ PIC16C621A	18P, 18JW 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XC0 PCM16XC0 PCM16XC0	DVA16XP180 DVA16XP180 DVA16XP180	— XLT18SO XLT20SS
PIC16C622/ PIC16C622A	18P, 18JW 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XC0 PCM16XC0 PCM16XC0	DVA16XP180 DVA16XP180 DVA16XP180	— XLT18SO XLT20SS
PIC16CE623	18P, 18JW 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XC0 PCM16XC0 PCM16XC0	DVA16XP180 DVA16XP180 DVA16XP180	— XLT18SO XLT20SS
PIC16CE624	18P, 18JW 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XC0 PCM16XC0 PCM16XC0	DVA16XP180 DVA16XP180 DVA16XP180	— XLT18SO XLT20SS
PIC16CE625	18P, 18JW 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XC0 PCM16XC0 PCM16XC0	DVA16XP180 DVA16XP180 DVA16XP180	— XLT18SO XLT20SS
PIC16C63	28SP, 28JW 28SO	ICE2000 ICE2000	PCM16XB1 PCM16XB1	DVA16XP281 DVA16XP281	— XLT28SO
PIC16C63A	28SP, 28JW 28SO 28SS	ICE2000 ICE2000 ICE2000	PCM16XE1 PCM16XE1 PCM16XE1	DVA16XP281 DVA16XP281 DVA16XP281	— XLT28SO XLT28SS
PIC16C64A	40P, 40JW 44L 44PQ, 44PT	ICE2000 ICE2000 ICE2000	PCM16XB1 PCM16XB1 PCM16XB1	DVA16XP400 DVA16XL440 DVA16PQ440	— — XLT44PT
PIC16C642	28SP, 28JW 28SO	ICE2000 ICE2000	PCM16XD0 PCM16XD0	DVA16XP281 DVA16XP281	— XLT28SO
PIC16C65A	40P, 40JW 44L 44PQ, 44PT	ICE2000 ICE2000 ICE2000	PCM16XB1 PCM16XB1 PCM16XB1	DVA16XP400 DVA16XL440 DVA16PQ440	— — XLT44PT
PIC16C65B	40P, 40JW 44L 44PQ, 44PT	ICE2000 ICE2000 ICE2000	PCM16XE1 PCM16XE1 PCM16XE1	DVA16XP400 DVA16XL440 DVA16PQ440	— — XLT44PT
PIC16C66	28SP, 28JW 28SO	ICE2000 ICE2000	PCM16XE1 PCM16XE1	DVA16XP281 DVA16XP281	— XLT28SO
PIC16C662	40P, 40JW 44L 44PQ, 44PT	ICE2000 ICE2000 ICE2000	PCM16XD0 PCM16XD0 PCM16XD0	DVA16XP400 DVA16XL440 DVA16PQ440	— — XLT44PT
PIC16C67	40P, 40JW 44L 44PQ, 44PT	ICE2000 ICE2000 ICE2000	PCM16XE1 PCM16XE1 PCM16XE1	DVA16XP400 DVA16XL440 DVA16PQ440	— — XLT44PT
PIC16C71	18P, 18JW 18SO	ICE2000 ICE2000	PCM16XF0 PCM16XF0	DVA16XP180 DVA16XP180	— XLT18SO
PIC16C710	18P, 18JW 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XF0 PCM16XF0 PCM16XF0	DVA16XP180 DVA16XP180 DVA16XP180	— XLT18SO XLT20SS

* Contact Microchip Technology Inc. for availability.

MPLAB™-ICE Cross Reference Parts List (Continued)

Model Name/ Part Number	Lead Count/ Package Type	Hardware Tools			
		Emulator Pod	Processor Module	Device Adapter	Transition Socket
PIC16C711	18P, 18JW 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XF0 PCM16XF0 PCM16XF0	DVA16XP180 DVA16XP180 DVA16XP180	— XLT18SO XLT20SS
PIC16C712(*)	18P, 18JW 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XE1 PCM16XE1 PCM16XE1	DVA16XP182 DVA16XP182 DVA16XP182	— XLT18SO XLT20SS
PIC16C715	18P, 18JW 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XG0 PCM16XG0 PCM16XG0	DVA16XP180 DVA16XP180 DVA16XP180	— XLT18SO XLT20SS
PIC16C716(*)	18P, 18JW 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XE1 PCM16XE1 PCM16XE1	DVA16XP182 DVA16XP182 DVA16XP182	— XLT18SO XLT20SS
PIC16C72	28SP, 28JW 28SO 28SS	ICE2000 ICE2000 ICE2000	PCM16XB1 PCM16XB1 PCM16XB1	DVA16XP281 DVA16XP281 DVA16XP281	— XLT28SO XLT28SS
PIC16C72A	28SP, 28JW 28SO 28SS	ICE2000 ICE2000 ICE2000	PCM16XE1 PCM16XE1 PCM16XE1	DVA16XP281 DVA16XP281 DVA16XP281	— XLT28SO XLT28SS
PIC16C73A	28SP, 28JW 28SO	ICE2000 ICE2000	PCM16XB1 PCM16XB1	DVA16XP281 DVA16XP281	— XLT28SO
PIC16C73B	28SP, 28JW 28SO 28SS	ICE2000 ICE2000 ICE2000	PCM16XE1 PCM16XE1 PCM16XE1	DVA16XP281 DVA16XP281 DVA16XP281	— XLT28SO XLT28SS
PIC16C74A	40P, 40JW 44L 44PQ, 44PT	ICE2000 ICE2000	PCM16XB1 PCM16XB1 PCM16XB1	DVA16XP400 DVA16XL440 DVA16PQ440	— — XLT44PT
PIC16C74B	40P, 40JW 44L 44PQ, 44PT	ICE2000 ICE2000 ICE2000	PCM16XE1 PCM16XE1 PCM16XE1	DVA16XP400 DVA16XL440 DVA16PQ440	— — XLT44PT
PIC16C76	28SP, 28JW 28SO	ICE2000 ICE2000	PCM16XE1 PCM16XE1	DVA16XP281 DVA16XP281	— XLT28SO
PIC16C77	40P, 40JW 44L 44PQ, 44PT	ICE2000 ICE2000 ICE2000	PCM16XE1 PCM16XE1 PCM16XE1	DVA16XP400 DVA16XL440 DVA16PQ440	— — XLT44PT
PIC16C773(*)	28SP, 28JW 28SO 28SS	ICE2000 ICE2000 ICE2000	PCM16XL0 PCM16XL0 PCM16XL0	DVA16XP281 DVA16XP281 DVA16XP281	— XLT28SO XLT28SS
PIC16C774(*)	40P, 40JW 44L 44PQ, 44PT	ICE2000 ICE2000 ICE2000	PCM16XL0 PCM16XL0 PCM16XL0	DVA16XP400 DVA16XL440 DVA16PQ440	— — XLT44PT
PIC16F83	18P 18SO	ICE2000 ICE2000	PCM16XH0 PCM16XH0	DVA16XP180 DVA16XP180	— XLT18SO
PIC16F84	18P 18SO	ICE2000 ICE2000	PCM16XH0 PCM16XH0	DVA16XP180 DVA16XP180	— XLT18SO
PIC16F84A	18P 18SO 20SS	ICE2000 ICE2000 ICE2000	PCM16XH0 PCM16XH0 PCM16XH0	DVA16XP180 DVA16XP180 DVA16XP180	— XLT18SO XLT20SS
PIC16F873*	28SP 28SO	ICE2000 ICE2000	PCM16XK0 PCM16XK0	DVA16XP281 DVA16XP281	— XLT28SO
PIC16F874(*)	40P 40L 44PQ, 44PT	ICE2000 ICE2000 ICE2000	PCM16XK0 PCM16XK0 PCM16XK0	DVA16XP400 DVA16XL440 DVA16PQ440	— — XLT44PT
PIC16F876(*)	28SP 28SO	ICE2000 ICE2000	PCM16XK0 PCM16XK0	DVA16XP281 DVA16XP281	— XLT28SO

* Contact Microchip Technology Inc. for availability.

MPLAB™-ICE Cross Reference Parts List (Continued)

Model Name/ Part Number	Lead Count/ Package Type	Hardware Tools			
		Emulator Pod	Processor Module	Device Adapter	Transition Socket
PIC16F877(*)	40P 40L 44PQ, 44PT	ICE2000 ICE2000 ICE2000	PCM16XK0 PCM16XK0 PCM16XK0	DVA16XP400 DVA16XL440 DVA16PQ440	— — XLT44PT
PIC16C923	64SP 64PT	ICE2000 ICE2000	PCM16XJ0 PCM16XJ0	DVA16XP640 DVA16PQ640	— XLT64PT1
PIC16C924	64SP 68L, 68CL 64PT	ICE2000 ICE2000 ICE2000	PCM16XJ0 PCM16XJ0 PCM16XJ0	DVA16XP640 DVA16XL680 DVA16PQ640	— — XLT64PT1
PIC17C42A	40P, 40JW 44L 44PQ, 44PT	ICE2000 ICE2000 ICE2000	PCM17XA0 PCM17XA0 PCM17XA0	DVA17XP400 DVA17XL440 DVA17PQ440	— — XLT44PT
PIC17C43	40P, 40JW 44L 44PQ, 44PT	ICE2000 ICE2000 ICE2000	PCM17XA0 PCM17XA0 PCM17XA0	DVA17XP400 DVA17XL440 DVA17PQ440	— — XLT44PT
PIC17C44	40P, 40JW 44L 44PQ, 44PT	ICE2000 ICE2000 ICE2000	PCM17XA0 PCM17XA0 PCM17XA0	DVA17XP400 DVA17XL440 DVA17PQ440	— — XLT44PT
PIC17C752	68L 64PT	ICE2000 ICE2000	PCM17XA0 PCM17XA0	DVA17XL680 DVA17PQ640	— XLT64PT2
PIC17C756/ PIC17C756A	68L, 68CL 64PT	ICE2000 ICE2000	PCM17XA0 PCM17XA0	DVA17XL680 DVA17PQ640	— XLT64PT2
PIC17C762	84L 80PT	ICE2000 ICE2000	PCM17XA0 PCM17XA0	DVA17XL840 DVA17PQ800	— XLT80PT
PIC17C766	84L, 84CL 80PT	ICE2000 ICE2000	PCM17XA0 PCM17XA0	DVA17XL840 DVA17PQ800	— XLT80PT
PIC18C242(*)	28P, 28JW 28SO	ICE2000 ICE2000	PCM18XA0 PCM18XA0	DVA18XP280 DVA18XP280	— XLT28SO
PIC18C252(*)	28P, 28JW 28SO	ICE2000 ICE2000	PCM18XA0 PCM18XA0	DVA18XP280 DVA18XP280	— XLT28SO
PIC18C442(*)	40P, 40JW 40L 44PT	ICE2000 ICE2000 ICE2000	PCM18XA0 PCM18XA0 PCM18XA0	DVA18XP400 DVA18XL440 DVA18PQ440	— — XLT44PT
PIC18C452(*)	40P, 40JW 40L 44PT	ICE2000 ICE2000 ICE2000	PCM18XA0 PCM18XA0 PCM18XA0	DVA18XP400 DVA18XL440 DVA18PQ440	— — XLT44PT

* Contact Microchip Technology Inc. for availability.

PICMASTER® Emulator Systems Cross Reference and Ordering Information

Model Name/ Part Number	PICMASTER POD	PICMASTER Probe Kit	PICMASTER-CE POD	PICMASTER-CE Probe Kit
PIC12C508(*)	AC008001	AC165004	EM007101	AC165015
PIC12C509(*)	AC008001	AC165004	EM007101	AC165015
PIC14000	AC008001	AC145001	EM007101	AC145002
PIC16C52	AC008001	AC165004	EM007101	AC165015
PIC16C54	AC008001	AC165004	EM007101	AC165015
PIC16C54A	AC008001	AC165004	EM007101	AC165015
PIC16C55	AC008001	AC165004	EM007101	AC165015
PIC16C554	AC008001	AC165030	EM007101	AC165020
PIC16C558	AC008001	AC165030	EM007101	AC165020
PIC16C56	AC008001	AC165004	EM007101	AC165015
PIC16C57	AC008001	AC165004	EM007101	AC165015
PIC16C58A	AC008001	AC165004	EM007101	AC165015
PIC16C620	AC008001	AC165008	EM007101	AC165018
PIC16C621	AC008001	AC165008	EM007101	AC165018
PIC16C622	AC008001	AC165008	EM007101	AC165018
PIC16C62A	AC008001	AC165009	EM007101	AC165016
PIC16C63	AC008001	AC165009	EM007101	AC165016
PIC16C642	AC008001	AC165031	EM007101	AC165021
PIC16C64A	AC008001	AC165009	EM007101	AC165016
PIC16C65A	AC008001	AC165009	EM007101	AC165016
PIC16C66	AC008001	AC165034	EM007101	AC165024
PIC16C662	AC008001	AC165031	EM007101	AC165021
PIC16C67	AC008001	AC165034	EM007101	AC165024
PIC16C71	AC008001	AC165010	EM007101	AC165013
PIC16C710	AC008001	AC165010	EM007101	AC165013
PIC16C711	AC008001	AC165010	EM007101	AC165013
PIC16C715	AC008001	AC165032	EM007101	AC165022
PIC16C72	AC008001	AC165009	EM007101	AC165016
PIC16C73A	AC008001	AC165009	EM007101	AC165016
PIC16C74A	AC008001	AC165009	EM007101	AC165016
PIC16C76	AC008001	AC165034	EM007101	AC165024
PIC16C77	AC008001	AC165034	EM007101	AC165024
PIC16C923	AC008001	AC165012	EM007101	AC165019
PIC16C924	AC008001	AC165012	EM007101	AC165019
PIC16F83	AC008001	AC165011	EM007101	AC165014
PIC16F84	AC008001	AC165011	EM007101	AC165014
PIC17C42A	AC008001	AC175002	EM007101	AC175003
PIC17C43	AC008001	AC175002	EM007101	AC175003
PIC17C44	AC008001	AC175002	EM007101	AC175003
PIC17C756	AC008001	AC175004	EM007101	AC175005

* PICMASTER PIC12CXXX emulation support also requires the use of a probe kit daughter board AC122001.

Model Name/ Part Number	ICEPIC Pod	ICEPIC Daughter Board
PIC12C508(*)	EM167200	AC165201
PIC12C509(*)	EM167200	AC165201
PIC16C52	EM167200	AC165201
PIC16C54	EM167200	AC165201
PIC16C54A	EM167200	AC165201
PIC16C55	EM167200	AC165201
PIC16C554	EM167200	AC165208
PIC16C558	EM167200	AC165208
PIC16C56	EM167200	AC165201
PIC16C57	EM167200	AC165201
PIC16C58A	EM167200	AC165201
PIC16C61	EM167200	AC165211
PIC16C620	EM167200	AC165202
PIC16C621	EM167200	AC165202
PIC16C622	EM167200	AC165202
PIC16C62A	EM167200	AC165207
PIC16C63	EM167200	AC165207
PIC16C642	EM167200	EM167213
PIC16C64A	EM167200	AC165207
PIC16C65A	EM167200	AC165207
PIC16C66	EM167200	AC165214
PIC16C662	EM167200	EM165213
PIC16C67	EM167200	AC165214
PIC16C71	EM167200	AC167211
PIC16C710	EM167200	AC167211
PIC16C711	EM167200	AC167211
PIC16C715	EM167200	AC167215
PIC16C72	EM167200	AC165207
PIC16C73A	EM167200	AC165207
PIC16C74A	EM167200	AC165207
PIC16C76	EM167200	AC165214
PIC16C77	EM167200	AC165214
PIC16C923	EM167200	AC165210
PIC16C924	EM167200	AC165210
PIC16F83	EM167200	AC165212
PIC16F84	EM167200	AC165212

* PIC12CXXX emulation support also requires the use of a kit daughter board adapter AC122002.

Software Tools Cross Reference and Ordering Information

Model Name/ Part Number	MPLAB™	MPLAB-C17	MPLAB-C18	Total Endurance™ Software Model	KEELOQ® License Disk
24CXX/24LCXX	—	—	—	SW242001	DS40149
93CXX/93LCXX	—	—	—	SW242001	DS40149
HCS200	—	—	—	—	DS40149
HCS201	—	—	—	—	DS40149
HCS300	—	—	—	—	DS40149
HCS301	—	—	—	—	DS40149
HCS320	—	—	—	—	DS40149
HCS360	—	—	—	—	DS40149
HCS361	—	—	—	—	DS40149
HCS410	—	—	—	—	DS40149
HCS412	—	—	—	—	DS40149
HCS500	—	—	—	—	DS40149
HCS512	—	—	—	—	DS40149
HCS515	—	—	—	—	DS40149
PIC12C508	SW007002	—	—	—	—
PIC12C508A	SW007002	—	—	—	—
PIC12C509	SW007002	—	—	—	—
PIC12C509A	SW007002	—	—	—	—
PIC12CE18	SW007002	—	—	—	—
PIC12CE19	SW007002	—	—	—	—
PIC12C671	SW007002	—	—	—	—
PIC12C672	SW007002	—	—	—	—
PIC12CE673	SW007002	—	—	—	—
PIC12CE674	SW007002	—	—	—	—
PIC14C000	SW007002	—	—	—	—
PIC16C505	SW007002	—	—	—	—
PIC16C52	SW007002	—	—	—	—
PIC16C54	SW007002	—	—	—	—
PIC16C54A	SW007002	—	—	—	—
PIC16C54C	SW007002	—	—	—	—
PIC16C55	SW007002	—	—	—	—
PIC16C55A	SW007002	—	—	—	—
PIC16C554	SW007002	—	—	—	—
PIC16C558	SW007002	—	—	—	—
PIC16C56	SW007002	—	—	—	—
PIC16C56A	SW007002	—	—	—	—
PIC16C57	SW007002	—	—	—	—
PIC16C57C	SW007002	—	—	—	—
PIC16C58A	SW007002	—	—	—	—
PIC16C58B	SW007002	—	—	—	—
PIC16C62A	SW007002	—	—	—	—
PIC16C62B	SW007002	—	—	—	—
PIC16C620	SW007002	—	—	—	—
PIC16C620A	SW007002	—	—	—	—
PIC16C621	SW007002	—	—	—	—
PIC16C621A	SW007002	—	—	—	—
PIC16C622	SW007002	—	—	—	—
PIC16C622A	SW007002	—	—	—	—
PIC16CE623	SW007002	—	—	—	—
PIC16CE624	SW007002	—	—	—	—

Shaded area indicates not applicable.

* Contact Microchip Technology Inc. for availability.

Software Tools Cross Reference and Ordering Information (Continued)

Model Name/ Part Number	MPLAB™	MPLAB-C17	MPLAB-C18	Total Endurance™ Software Model	KEELOQ® License Disk
PIC16CE625	SW007002	—	—	—	—
PIC16F627(*)	SW007002	—	—	—	—
PIC16F628(*)	SW007002	—	—	—	—
PIC16C63	SW007002	—	—	—	—
PIC16C63A	SW007002	—	—	—	—
PIC16C64A	SW007002	—	—	—	—
PIC16C642	SW007002	—	—	—	—
PIC16C65A	SW007002	—	—	—	—
PIC16C65B	SW007002	—	—	—	—
PIC16C66	SW007002	—	—	—	—
PIC16C662	SW007002	—	—	—	—
PIC16C67	SW007002	—	—	—	—
PIC16C71	SW007002	—	—	—	—
PIC16C710	SW007002	—	—	—	—
PIC16C711	SW007002	—	—	—	—
PIC16C712	SW007002	—	—	—	—
PIC16C715	SW007002	—	—	—	—
PIC16C716	SW007002	—	—	—	—
PIC16C717	SW007002	—	—	—	—
PIC16C72	SW007002	—	—	—	—
PIC16C72A	SW007002	—	—	—	—
PIC16C73B	SW007002	—	—	—	—
PIC16C73C(*)	SW007002	—	—	—	—
PIC16C74A	SW007002	—	—	—	—
PIC16C74B	SW007002	—	—	—	—
PIC16C76	SW007002	—	—	—	—
PIC16C77	SW007002	—	—	—	—
PIC16C770(*)	SW007002	—	—	—	—
PIC16C771(*)	SW007002	—	—	—	—
PIC16C773	SW007002	—	—	—	—
PIC16C774	SW007002	—	—	—	—
PIC16F83	SW007002	—	—	—	—
PIC16F84	SW007002	—	—	—	—
PIC16F84A	SW007002	—	—	—	—
PIC16F873	SW007002	—	—	—	—
PIC16F874	SW007002	—	—	—	—
PIC16F876	SW007002	—	—	—	—
PIC16F877	SW007002	—	—	—	—
PIC16C923	SW007002	—	—	—	—
PIC16C924	SW007002	—	—	—	—
PIC17C42A	SW007002	SW006010	—	—	—
PIC17C43	SW007002	SW006010	—	—	—
PIC17C44	SW007002	SW006010	—	—	—
PIC17C752	SW007002	SW006010	—	—	—
PIC17C756	SW007002	SW006010	—	—	—
PIC17C756A	SW007002	SW006010	—	—	—
PIC17C762	SW007002	SW006010	—	—	—
PIC17C766	SW007002	SW006010	—	—	—
PIC18C242(*)	SW007002	—	SW006011(*)	—	—
PIC18C252(*)	SW007002	—	SW006011(*)	—	—

Shaded area indicates not applicable.

* Contact Microchip Technology Inc. for availability.

Software Tools Cross Reference and Ordering Information (Continued)

Model Name/ Part Number	MPLAB™	MPLAB-C17	MPLAB-C18	Total Endurance™ Software Model	KEELOQ® License Disk
PIC18C442(*)	SW007002	—	SW006011(*)	—	—
PIC18C452(*)	SW007002	—	SW006011(*)	—	—

Shaded area indicates not applicable.

* Contact Microchip Technology Inc. for availability.

Programmers Cross Reference and Ordering Information

Model Name/ Part Number	PICSTART Plus	PRO MATE II	PRO MATE II						
			DIP	SOIC	SSOP	PLCC	MQFP	TQFP	ICSP Module
24CXX/ 24LCXX	—	DV007003	AC004001	AC004002	—	—	—	—	AC004004
93CXX/ 93LCXX	—	DV007003	AC004001	AC004002	—	—	—	—	AC004004
HCS200	—	DV007003	AC004001	AC004002	—	—	—	—	AC004004
HCS201	—	DV007003	AC004001	AC004002	—	—	—	—	AC004004
HCS300	—	DV007003	AC004001	AC004002	—	—	—	—	AC004004
HCS301	—	DV007003	AC004001	AC004002	—	—	—	—	AC004004
HCS320	—	DV007003	AC004001	AC004002	—	—	—	—	AC004004
HCS360	—	DV007003	AC004001	AC004002	—	—	—	—	AC004004
HCS361	—	DV007003	AC004001	AC004002	—	—	—	—	AC004004
HCS410	—	DV007003	AC004001	AC004002	—	—	—	—	AC004004
HCS412	—	DV007003	AC004001	AC004002	—	—	—	—	AC004004
HCS500(*)	—	DV007003	—	—	—	—	—	—	AC004004
HCS512	—	DV007003	AC164001	AC164002	—	—	—	—	AC004004
HCS515(*)	—	DV007003	—	—	—	—	—	—	AC004004
PIC12C508	DV003001	DV007003	AC124001	AC124001	—	—	—	—	AC004004
PIC12C508A	DV003001	DV007003	AC124001	AC124001(†) AC164026(†)	—	—	—	—	AC004004
PIC12C509	DV003001	DV007003	AC124001	AC124001	—	—	—	—	AC004004
PIC12C509A	DV003001	DV007003	AC124001	AC124001(†) AC164026(†)	—	—	—	—	AC004004
PIC12CE518	DV003001	DV007003	AC124001	AC124001(†) AC164026(†)	AC164026	—	—	—	AC004004
PIC12CE519	DV003001	DV007003	AC124001	AC124001(†) AC164026(†)	AC164026	—	—	—	AC004004
PIC12C671	DV003001	DV007003	AC124001	AC124001	—	—	—	—	AC004004
PIC12C672	DV003001	DV007003	AC124001	AC124001	—	—	—	—	AC004004
PIC12CE673	DV003001	DV007003	AC124001	—	—	—	—	—	AC004004
PIC12CE674	DV003001	DV007003	AC124001	—	—	—	—	—	AC004004
PIC14C000	DV003001	DV007003	AC144001	AC144002	AC144002	—	—	—	AC004004
PIC16C505	DV003001	DV007003	AC124001	AC164026	—	—	—	—	AC004004
PIC16C52	DV003001	DV007003	AC164001	AC164002	—	—	—	—	AC004004
PIC16C54	DV003001	DV007003	AC164001	AC164002	AC164015	—	—	—	AC004004
PIC16C54A	DV003001	DV007003	AC164001	AC164002	AC164015	—	—	—	AC004004
PIC16C54C	DV003001	DV007003	AC164001	AC164002	AC164015	—	—	—	AC004004
PIC16HV540	DV003001	DV007003	AC164001	AC164002	AC164015				AC004004
PIC16C55	DV003001	DV007003	AC164001	AC164002	AC164015	—	—	—	AC004004
PIC16C55A	DV003001	DV007003	AC164001	AC164002	AC164015	—	—	—	AC004004
PIC16C554	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16C558	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16C56	DV003001	DV007003	AC164001	AC164002	AC164015	—	—	—	AC004004
PIC16C56A	DV003001	DV007003	AC164001	AC164002	AC164015	—	—	—	AC004004
PIC16C57	DV003001	DV007003	AC164001	AC164002	AC164015	—	—	—	AC004004
PIC16C57C	DV003001	DV007003	AC164001	AC164002	AC164015	—	—	—	AC004004
PIC16C58A	DV003001	DV007003	AC164001	AC164002	AC164015	—	—	—	AC004004

Shaded area indicates not applicable.

* Contact Microchip Technology Inc. for availability.

† AC124001 (208 mil); AC164026 (150 mil).

Programmers Cross Reference and Ordering Information (Continued)

Model Name/ Part Number	PICSTART Plus	PRO MATE II	PRO MATE II						
			DIP	SOIC	SSOP	PLCC	MQFP	TQFP	ICSP Module
PIC16C58B	DV003001	DV007003	AC164001	AC164002	AC164015	—	—	—	AC004004
PIC16C62A	DV003001	DV007003	AC164012	AC164017	AC164021	—	—	—	AC004004
PIC16C62B	DV003001	DV007003	AC164012	AC164017	AC164021	—	—	—	AC004004
PIC16C620	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16C620A	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16C621	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16C621A	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16C622	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16C622A	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16CE623	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16CE624	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16CE625	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16F627(*)	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16F628(*)	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16C63	DV003001	DV007003	AC164012	AC164017	—	—	—	—	AC004004
PIC16C63A	DV003001	DV007003	AC164012	AC164017	AC164021	—	—	—	AC004004
PIC16C64A	DV003001	DV007003	AC164012	—	—	AC164013	AC164014	AC164020	AC004004
PIC16C642	DV003001	DV007003	AC164012	AC164017	—	—	—	—	AC004004
PIC16C65A	DV003001	DV007003	AC164012	—	—	AC164013	AC164014	AC164020	AC004004
PIC16C65B	DV003001	DV007003	AC164012	—	—	AC164013	AC164014	AC164020	AC004004
PIC16C66	DV003001	DV007003	AC164012	AC164017	—	—	—	—	AC004004
PIC16C662	DV003001	DV007003	AC164012	—	—	AC164013	AC164014	AC164020	AC004004
PIC16C67	DV003001	DV007003	AC164012	—	—	AC164013	AC164014	AC164020	AC004004
PIC16C71	DV003001	DV007003	AC164010	AC164010	—	—	—	—	AC004004
PIC16C710	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16C711	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16C712(*)	DV003001	DV007003	AC164010	AC164010	AC164018				AC004004
PIC16C715	DV003001	DV007003	AC164010	AC164010	AC164018	—	—	—	AC004004
PIC16C716(*)	DV003001	DV007003	AC164010	AC164010	AC164018				AC004004
PIC16C717(*)	DV003001	DV007003	AC164010	AC164010	AC164018				AC004004
PIC16C72	DV003001	DV007003	AC164012	AC164017	AC164021	—	—	—	AC004004
PIC16C72A	DV003001	DV007003	AC164012	AC164017	AC164021	—	—	—	AC004004
PIC16C73A	DV003001	DV007003	AC164012	AC164017	—	—	—	—	AC004004
PIC16C73B	DV003001	DV007003	AC164012	AC164017	AC164021	—	—	—	AC004004
PIC16C74A	DV003001	DV007003	AC164012	—	—	AC164013	AC164014	AC164020	AC004004
PIC16C74B	DV003001	DV007003	AC164012	—	—	AC164013	AC164014	AC164020	AC004004
PIC16C76	DV003001	DV007003	AC164012	AC164017	—	—	—	—	AC004004
PIC16C77	DV003001	DV007003	AC164012	—	—	AC164013	AC164014	AC164020	AC004004
PIC16C770(*)	DV003001	DV007003	AC164028	AC164028	AC164018	—	—	—	AC004004
PIC16C771(*)	DV003001	DV007003	AC164028	AC164028	AC164018	—	—	—	AC004004
PIC16C773	DV003001	DV007003	AC164012	AC164017	AC164021	—	—	—	AC004004
PIC16C774	DV003001	DV007003	AC164012	—	—	AC164013	AC164014	AC164020	AC004004
PIC16F83	DV003001	DV007003	AC164010	AC164010	—	—	—	—	AC004004
PIC16F84	DV003001	DV007003	AC164010	AC164010	—	—	—	—	AC004004
PIC16F84A	DV003001	DV007003	AC164010	AC164010	—	—	—	—	AC004004
PIC16F873	DV003001	DV007003	AC164012	AC164017	—	—	—	—	AC004004
PIC16F874	DV003001	DV007003	AC164012	—	—	AC164013	AC164014	AC164020	AC004004
PIC16F876	DV003001	DV007003	AC164012	AC164017	—	—	—	—	AC004004
PIC16F877	DV003001	DV007003	AC164012	—	—	AC164013	AC164014	AC164020	AC004004
PIC16C923	DV003001	DV007003	AC164025	—	—	—	—	AC164023	AC004004
PIC16C924	DV003001	DV007003	AC164025	—	—	AC164022	—	AC164023	AC004004
PIC17C42A	DV003001	DV007003	AC174001	—	—	AC174002	AC174004	AC174005	AC004004
PIC17C43	DV003001	DV007003	AC174001	—	—	AC174002	AC174004	AC174005	AC004004
PIC17C44	DV003001	DV007003	AC174001	—	—	AC174002	AC174004	AC174005	AC004004

Shaded area indicates not applicable.

* Contact Microchip Technology Inc. for availability.

† AC124001 (208 mil); AC164026 (150 mil).

Programmers Cross Reference and Ordering Information (Continued)

Model Name/ Part Number	PICSTART Plus	PRO MATE II	PRO MATE II						
			DIP	SOIC	SSOP	PLCC	MQFP	TQFP	ICSP Module
PIC17C752	DV003001	DV007003	—	—	—	AC174007	—	AC174008	AC004004
PIC17C756	DV003001	DV007003	—	—	—	AC174007	—	AC174008	AC004004
PIC17C756A	DV003001	DV007003	—	—	—	AC174007	—	AC174008	AC004004
PIC17C762	DV003001	DV007003	—	—	—	AC174012	—	AC174011	AC004004
PIC17C766	DV003001	DV007003	—	—	—	AC174012	—	AC174011	AC004004
PIC18C242(*)	DV003001	DV007003	AC164012	AC164017	—	—	—	—	AC004004
PIC18C252(*)	DV003001	DV007003	AC164012	AC164017	—	—	—	—	AC004004
PIC18C442(*)	DV003001	DV007003	AC164012	—	—	AC164013	—	AC164020	AC004004
PIC18C452(*)	DV003001	DV007003	AC164012	—	—	AC164013	—	AC164020	AC004004

Shaded area indicates not applicable.
* Contact Microchip Technology Inc. for availability.
† AC124001 (208 mil); AC164026 (150 mil).

PICmicro MCU Demo Boards and Evaluation Kits Cross Reference and Ordering Information

Model Name/ Part Number	SIMICE	PICDEM-1	PICDEM-2	PICDEM-3	PICDEM-14A	PICDEM-17
PIC12C508	DV162010	DM163001	—	—	—	—
PIC12C508A	DV162010	DM163001	—	—	—	—
PIC12C509	DV162010	DM163001	—	—	—	—
PIC12C509A	DV162010	DM163001	—	—	—	—
PIC12CE518	—	—	—	—	—	—
PIC12CE519	—	—	—	—	—	—
PIC12C671	—	—	—	—	—	—
PIC12C672	—	—	—	—	—	—
PIC12CE673	—	—	—	—	—	—
PIC12CE674	—	—	—	—	—	—
PIC14C000	—	—	—	—	DM143001	—
PIC16C505	—	—	—	—	—	—
PIC16C52	DV162010	DM163001	—	—	—	—
PIC16C54	DV162010	DM163001	—	—	—	—
PIC16C54A	DV162010	DM163001	—	—	—	—
PIC16C54C	DV162010	DM163001	—	—	—	—
PIC16HV540	—	—				
PIC16C55	DV162010	DM163001	—	—	—	—
PIC16C55A	DV162010	DM163001	—	—	—	—
PIC16C554	—	DM163001	—	—	—	—
PIC16C558	—	DM163001	—	—	—	—
PIC16C56	DV162010	DM163001	—	—	—	—
PIC16C56A	DV162010	DM163001	—	—	—	—
PIC16C57	DV162010	DM163001	—	—	—	—
PIC16C57C	DV162010	DM163001	—	—	—	—
PIC16C58A	DV162010	DM163001	—	—	—	—
PIC16C58B	DV162010	DM163001	—	—	—	—
PIC16C62A	—	—	DM163002	—	—	—
PIC16C62B	—	—	DM163002	—	—	—
PIC16C620	—	DM163001	—	—	—	—
PIC16C620A	—	DM163001	—	—	—	—
PIC16C621	—	DM163001	—	—	—	—
PIC16C621A	—	DM163001	—	—	—	—
PIC16C622	—	DM163001	—	—	—	—
PIC16C622A	—	DM163001	—	—	—	—
PIC16CE623	—	—	—	—	—	—
PIC16CE624	—	—	—	—	—	—
PIC16CE625	—	—	—	—	—	—
PIC16F627	—	—	—	—	—	—
PIC16F628	—	—	—	—	—	—
PIC16C63	—	—	DM163002	—	—	—
PIC16C63A	—	—	DM163002	—	—	—
PIC16C64A	—	—	DM163002	—	—	—
PIC16C642	—	—	DM163002	—	—	—
PIC16C65A	—	—	DM163002	—	—	—
PIC16C65B	—	—	DM163002	—	—	—
PIC16C66	—	—	DM163002	—	—	—
PIC16C662	—	—	DM163002	—	—	—
PIC16C67	—	—	DM163002	—	—	—
PIC16C71	—	DM163001	—	—	—	—
PIC16C710	—	DM163001	—	—	—	—
PIC16C711	—	DM163001	—	—	—	—
PIC16C712	—	—	—	—	—	—
PIC16C715	—	DM163001	—	—	—	—
PIC16C716	—	—	—	—	—	—

Shaded area indicates not applicable.

PICmicro MCU Demo Boards and Evaluation Kits Cross Reference and Ordering Information (Continued)

Model Name/ Part Number	SIMICE	PICDEM-1	PICDEM-2	PICDEM-3	PICDEM-14A	PICDEM-17
PIC16C717	—	—	—	—	—	—
PIC16C72	—	—	DM163002	—	—	—
PIC16C72A	—	—	DM163002	—	—	—
PIC16C73A	—	—	DM163002	—	—	—
PIC16C73B	—	—	DM163002	—	—	—
PIC16C74A	—	—	DM163002	—	—	—
PIC16C74B	—	—	DM163002	—	—	—
PIC16C76	—	—	DM163002	—	—	—
PIC16C77	—	—	DM163002	—	—	—
PIC16C773	—	—	—	—	—	—
PIC16C774	—	—	DM163002	—	—	—
PIC16F83	—	DM163001	—	—	—	—
PIC16F84	—	DM163001	—	—	—	—
PIC16F84A	—	DM163001	—	—	—	—
PIC16F873	—	—	DM163002	—	—	—
PIC16F874	—	—	DM163002	—	—	—
PIC16F876	—	—	DM163002	—	—	—
PIC16F877	—	—	DM163002	—	—	—
PIC16C923	—	—	—	DM163003	—	—
PIC16C924	—	—	—	DM163003	—	—
PIC17C42A	—	DM163001	—	—	—	—
PIC17C43	—	DM163001	—	—	—	—
PIC17C44	—	DM163001	—	—	—	—
PIC17C752	—	—	—	—	—	DM173001
PIC17C756	—	—	—	—	—	DM173001
PIC17C756A	—	—	—	—	—	DM173001
PIC17C762	—	—	—	—	—	DM173001
PIC17C766	—	—	—	—	—	DM173001
PIC18C242	—	—	DM163002	—	—	—
PIC18C252	—	—	DM163002	—	—	—
PIC18C442	—	—	DM163002	—	—	—
PIC18C452	—	—	DM163002	—	—	—

Shaded area indicates not applicable.

KEELOQ, microID™, and Serial EERPOM Evaluation Kits

Model Name/ Part No.	KEELOQ® Evaluation Kit	KEELOQ Transponder Evaluation Kit	Serial EEPROM Design Kit	MCP2510 CAN Development Kit*	microID Programmer Kit	125 kHz microID Developer's Kit	125 kHz Anticollision microID Developer's Kit	13.56 MHz Anticollision microID Developer's Kit
24CXX/ 24LCXX	—	—	DV243001	—	—	—	—	—
93CXX/ 93LCXX	—	—	DV243001	—	—	—	—	—
HCS200	DM303002	—	—	—	—	—	—	—
HCS201	DM303002	—	—	—	—	—	—	—
HCS300	DM303002	—	—	—	—	—	—	—
HCS301	DM303002	—	—	—	—	—	—	—
HCS320	DM303002	—	—	—	—	—	—	—
HCS360	DM303002	—	—	—	—	—	—	—
HCS361	DM303002	—	—	—	—	—	—	—
HCS410	—	DM303005	—	—	—	—	—	—
HCS412	—	DM303005	—	—	—	—	—	—
HCS500	DM303002	—	—	—	—	—	—	—
HCS512	DM303002	—	—	—	—	—	—	—
HCS515	DM303002	—	—	—	—	—	—	—
MCRF200	—	—	—	—	PG103001	DV103001	—	—
MCRF250	—	—	—	—	PG103001	—	DV103002	—
MCRF355	—	—	—	—	—	—	—	DV103003
MCP2510	—	—	—	DV251001	—	—	—	—

Shaded area indicates not applicable.

* Contact Microchip Technology Inc. for availability.

ON-LINE SUPPORT

Microchip Internet Connections

On-line Support

Microchip provides on-line support on the Microchip World Wide Web (WWW) site.

The web site is used by Microchip as a means to make files and information easily available to customers. To view the site, the user must have access to the internet and a web browser, such as Netscape® or Microsoft® Internet Explorer®. Files are also available for FTP download from our FTP site.

Connecting to the Microchip Internet Web Site

The Microchip web site is available by using your favorite internet browser to attach to:

www.microchip.com

The file transfer site is available by using an FTP service to connect to:

ftp://www.microchip.com

The web site and file transfer site provide a variety of services. Users may download files for the latest development tools, data sheets, application notes, user's guides, articles, and sample programs. A variety of Microchip specific business information is also available, including listings of Microchip sales offices, distributors and factory representatives. Other data available for consideration is:

- Latest Microchip press releases
- Technical support section with frequently asked questions
- Design tips
- Device errata
- Job postings
- Microchip consultant program member listing
- Links to other useful web sites related to Microchip products
- Conferences for products, development systems, technical information and more
- Listing of seminars and events

Systems Information and Upgrade Hot Line

The Systems Information and Upgrade Hot Line provides system users a listing of the latest versions of all of Microchip's development systems software products. Plus, this line provides information on how customers can receive any currently available upgrade kits. The Hot Line Numbers are:

1-800-755-2345 for U.S. and most of Canada

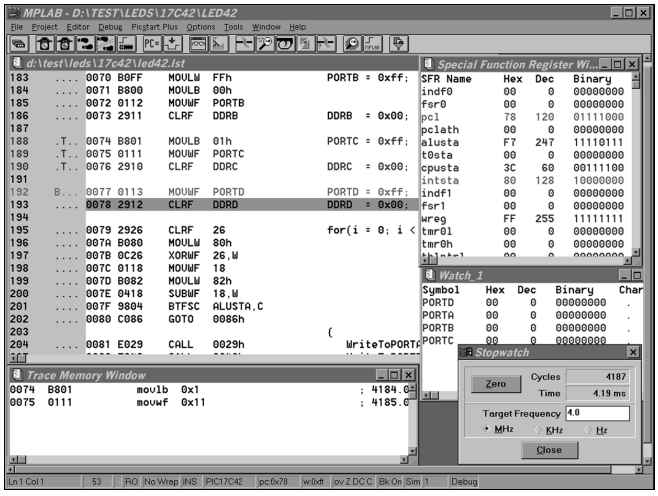
and

1-480-786-7302 for the rest of the world

NOTES:

MPLAB™

Integrated Development Environment



MPLAB gives PICmicro® MCU users the flexibility to edit, compile, and debug from a single user interface.

MPLAB is a Windows®-based development platform for the Microchip Technology PICmicro microcontroller (MCU) families. MPLAB Integrated Development Environment (IDE) offers a project manager and program text editor, a user-configurable toolbar containing four predefined sets, and a status bar which communicates editing and debugging information.

MPLAB is the common user interface for Microchip development systems tools including MPLAB Editor, MPASM Assembler, MPLAB-SIM Software Simulator, MPLIB, MPLINK, MPLAB-C17 C Compiler, MPLAB-ICE In-Circuit Emulator, PICSTART® Plus Development Programmer, and PRO MATE® II Programmer. Additional products may become available as add-on tools in the future.

The MPLAB desktop provides the development environment and tools for developing and debugging your application as a project, allowing you to quickly move between different development and debugging modes. With the MPLAB environment, you can write and debug your source code, automatically locate errors in source files for editing, debug with breakpoints based on internal register values, watch the program flow with MPLAB-SIM (software simulator) or MPLAB-ICE, make timing measurements with a "stop watch," view variables in watch windows, program firmware with PICSTART Plus or PRO MATE II programmers, and find quick answers to questions from the MPLAB on-line help.

Features:

MPLAB Project Manager

- Organizes the different files that comprise your application firmware under one "Project"
- Allows you to create a project; add, edit or debug a source code file in a project; build object files: and download code to the emulator or simulator with a mouse click
- Supports multiple source files such as MPASM and MPLAB-C17 source files, precompiled libraries and object files, and linker scripts
- Includes configurable build tools

MPLAB-SIM Software Simulator

- Discrete-event simulator integrated into the MPLAB IDE features debug capabilities including unlimited breakpoints, trace, examine/modify registers, watch variables, and time-stamp
- Simulates the core functions as well as most peripherals of the PICmicro MCUs

MPLAB Editor

- Full-featured programmer's editor lets you write and edit firmware source files or other text files for PICmicro MCUs
- Creates reusable source file templates for quick creation of new source files

MPASM Universal Assembler

- Has full-featured macro capabilities, conditional assembly and several source and listing formats
- Generates various object code formats to support Microchip development tools and third-party programmers without exiting MPLAB

MPLINK Linker

- A linker for Microchip's MPLAB-C17 C compiler and the relocatable assembler, MPASM
- Combines multiple input object modules, generated by MPLAB-C17 or MPASM, into a single executable file
- Generates the symbolic information for debugging with MPLAB

MPLAB

Ordering Information:

Model Name:

MPLAB IDE

Ordering Part Number:

SW007002

Tools Supported:

MPLAB-ICE In-Circuit Emulator

ICEPIC Emulator System

SIMICE Entry-level Hardware Simulator

PICSTART Plus Low-cost Programmer

PRO MATE II Full-featured Programmer

Host System Requirements:

PC with 386 or higher processor. Pentium® recommended

8 MB Memory, 32 MB recommended

16 MB hard disk space, 20 MB recommended

VGA or Super VGA Monitor

Microsoft® Windows 3.1 or greater

CD-ROM Drive

System Description:

MPLAB allows you to write, debug, and optimize the PICmicro MCU applications for firmware product designs.

MPLAB IDE software package includes the following:

- MPLAB Project Manager
- MPLAB-SIM Software Simulator
- MPLAB Editor
- MPASM Universal Macro Assembler for the PICmicros and other language products supporting the Common Object Description file Format
- MPLINK, MPLIB

To order or obtain more information about MPLAB IDE, contact the Microchip sales office nearest you.

Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers, and Corporate Application Engineers. Microchip's Internet home page can be reached at: www.microchip.com.

Development Tools from Microchip	
MPLAB™	Integrated Development Environment (IDE)
MPLAB-C17	C compiler for PIC17CXXX MCUs
MPLAB-C18	C compiler for PIC18CXXX MCUs
C compiler	Sold by third-party vendors (HI-TECH, IAR, CCS)
MPASM	Universal PICmicro macro-assembler
MPLINK/MPLIB	Linker/Librarian
MPLAB-SIM	Software Simulator
MPLAB-ICD	In-circuit debugger evaluation kit
MPLAB-ICE 2000	Full-featured modular in-circuit emulator
ICEPIC	Low-cost in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART®Plus	Entry-level development kit with programmer

MPASM

Universal PICmicro® Microcontroller Assembler Software

This product brief describes the technical aspects of the PICmicro MCU Assembler. The MPASM Cross Assembler is a PC hosted symbolic assembler. It supports all microcontroller series, including the PIC12CXX, PIC16C5X, PIC16CXX and PIC17CXX families.

MPASM offers fully featured Macro capabilities, conditional assembly, and several source and listing formats. It generates various object code formats to support Microchip's development tools as well as third party programmers.

MPASM allows full symbolic debugging from the MPLAB Integrated Development Environment.

MPASM REQUIREMENTS

MPASM will run on any IBM PC/AT® or compatible computer running DOS 5.0 or later.

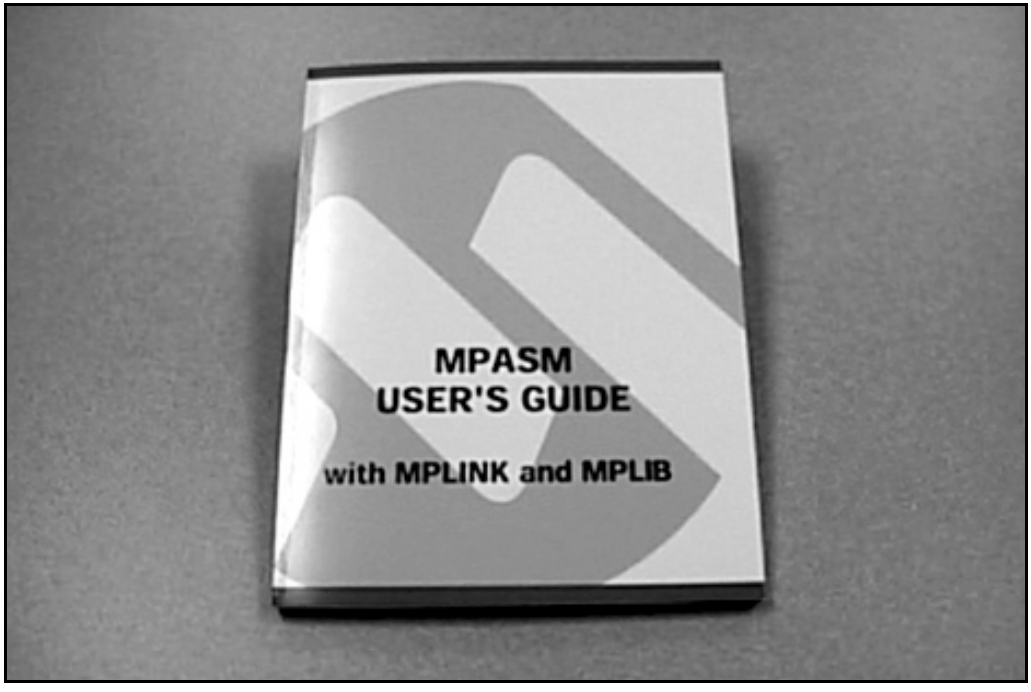
MPASM ASSEMBLER FEATURES

MPASM supports the 12-bit PIC12CXX and PIC16C5X, the 14-bit PIC16CXX, and the 16-bit PIC17CXX cores.

All instructions are single-word and single-cycle, except for branches, which execute in two cycles. Most instructions operate on one or more operands.

MPASM have the following features to assist in developing software for specific user applications:

- Provides translation of Assembler source code to object code for all Microchip microcontrollers.
- Macro Assembly Capability
- Provides Object, Listing, Symbol and special files required for debugging with one of the Microchip Emulator systems.
- Supports Hex (default), Decimal and Octal source and listing formats.
- Output formats: INHX8S, INHX8M, INHX32 and relocatable objects.



MPASM DIRECTIVE LANGUAGE

MPASM provides a full featured directive language represented by four basic classes of directives:

- Data Directives are those that control the allocation of memory and provide a way to refer to data items symbolically, by meaningful names.
- Listing Directives control the MPASM listing display. They allow the specification of titles and subtitles, page ejects and other listing control.
- Control Directives permit sections of conditionally assembled code.
- Macro Directives control the execution and data allocation within macro body definitions.

MPLINK

MPLINK provides a linker for relocatable objects produced by MPASM and MPLAB-C17. It allows the production of reusable code and can combine objects generated by both MPASM and MPLAB-C17 into an application

MPLINK is extremely flexible because it is controlled with a linker script. The script defines the processor ROM and RAM memory regions, and relocatable objects from the compiler and assembler are placed into available program memory locations. RAM is allocated as needed. If code space overflows the available ROM or if variables are used beyond the capacity of the current device, MPLINK will give an error message.

MPLINK allows blocks of data to be defined as reusable, so that routines that never call each other can make efficient use of RAM.

MPLINK generates map files to display usage of processor resources and locations of global variables and function executables. It also generates debugging files for use with MPLAB. Symbolic information is included in these files which allows MPLAB to track source lines, variables, and executable code.

MPLINK is distributed in two executable formats: a Win32 console application suitable for Windows 95 and Win NT platforms, and an extended DOS DPMI application suitable for Windows 3.x and DOS platforms.

MPLIB

MPLIB manages the creation and modification of library files. Library files have distinct advantages over re-coding routines in all applications:

- Libraries make linking easier – multiple objects can be contained in a single library.
- Libraries help keep code small – only the routines that are used in an application are extracted from the library.
- Libraries make projects more maintainable – deleting or adding calls to a library does not change the overall build process.

MPASM INSTRUCTION SET

MPASM supports the entire instruction set of the PIC16C5X, PIC16CXX and PIC17CXX microcontrollers, as represented in the following four classes of instructions:

- Data Move Operations
- Arithmetic and Logical Operations
- Bit Manipulation Operations
- Special Control Operations

The Microchip microcontroller set is used to operate on data located in any of the file registers, including the I/O registers. There are:

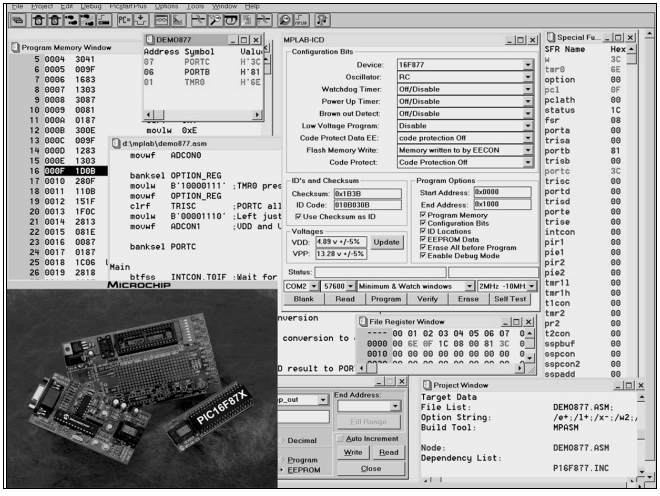
- Data Transfer Operations
- Logical Operations
- Rotate Operations

MPASM provides bit level file register operations to manipulate and test individual bits in any addressable register, literal and control operations permitting operations on literals and branches to subroutines in program memory.

The Microchip microcontroller instruction sets allow read and write of special function registers such as the PC and status registers.

MPLAB™-ICD

In-Circuit Debugger



Develop your next FLASH PICmicro® MCU project with MPLAB-ICD, a powerful and affordable development and evaluation kit!

Microchip's In-Circuit Debugger, MPLAB-ICD, is a powerful, low-cost development and evaluation kit for the FLASH PIC16F87X microcontroller (MCU) family. MPLAB-ICD utilizes the In-Circuit Debugging capability of the PIC16F87X. This feature, along with Microchip's In-Circuit Serial Programming™ (ICSP™) protocol, offers cost-effective

in-circuit FLASH programming and debugging from the graphical user interface of the MPLAB Integrated Development Environment (IDE). A designer can develop and debug source code by watching variables, setting break points, and single-stepping. Running at full speed enables testing hardware in real-time.

The modular design of the In-Circuit Debugger consists of three basic components: ICD module, ICD header, and ICD demo board. The ICD module connects to a serial (COM) port of a PC. When instructed by MPLAB IDE, the ICD module programs and issues debug commands to the target PIC16F87X using ICSP protocol. A 9-inch, 6 conductor cable connects the ICD module to the ICD header. The header contains a target PIC16F877, a modular jack, and 28-pin and 40-pin male DIP headers. The

28-pin and 40-pin DIP headers can be plugged into a target circuit board or into the ICD demo board. A modular jack can be designed into at a target circuit board to support direct connection to the ICD module or, alternatively, a DIP socket on a target application can support direct connection to the ICD header. If a target application is not available, immediate prototype development using the MPLAB-ICD is feasible with the included ICD demo board. This board offers LEDs, DIP switches, an analog potentiometer, and prototyping area.

The MPLAB-ICD complete hardware development system along with the free MPLAB software provides a powerful, affordable run-time development tool.

Features:

- PIC16F87X evaluation and demonstration board
- PIC16F87X device programming
- In-circuit run-time debugging
- Real-time code execution
- One hardware break point
- Single step
- Watch variables
- 3.0V to 5.5V operating
- 32 kHz to 20 MHz operation
- PC communication at speeds up to 57600 baud
- Includes Microchip's MPLAB IDE:
  - Editor
  - Assembler
  - Linker
  - Simulator
  - Project Manager
  - Source level symbolic debug

MPLAB™-ICD

Ordering Information:

MPLAB-ICD

Ordering Part Number:

DV164001

Devices Supported:

PIC16F873

PIC16F874

PIC16F876

PIC16F877

Contact Microchip Technology's web site at www.microchip.com for information on how to use the MPLAB-ICD with:

PIC16C62	PIC16C72
PIC16C63	PIC16C73
PIC16C64	PIC16C74
PIC16C65	PIC16C76
PIC16C66	PIC16C77

Host System Requirements:

PC-compatible machine with 486 or higher processor

4 MB RAM, 16 MB recommended
20 MB available hard disk space

Microsoft® Windows® 3.X/95/98

CD-ROM Drive

One free serial port

9V, 0.75A Power Supply
(PICSTART® Plus or equivalent)

System Description:

The low-cost PC- based MPLAB-ICD comes with the ICD module; ICD header; ICD demo board; RS-232 cable; 40-pin DIP and 28-pin SDIP connection sockets; one 9-inch, 6-conductor modular cable; MPLAB IDE software; and complete documentation. MPLAB-ICD requires the user to provide a power supply for operation. (A PICSTART® Plus or equivalent 9V, 0.75A power supply is required.)

The MPLAB-ICD module connects to the serial port of the host PC via the RS-232 cable. The 9-inch modular cable connects the MPLAB-ICD module to the

MPLAB-ICD header. The MPLAB-ICD header plugs into the connection socket located on the demo board or a target application. A user provided power supply from the demo board or target application, powers the MPLAB-ICD module.

To obtain more information about MPLAB-ICD or any other Microchip product, contact the Microchip sales office nearest you or visit the Microchip web site.

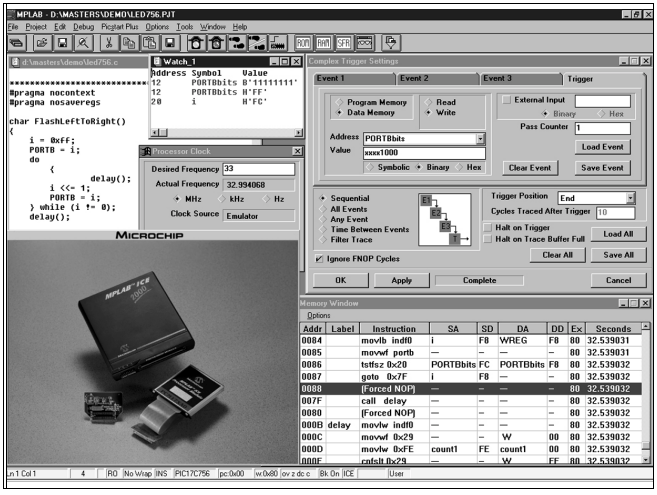
Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers, and Corporate Application Engineers. Microchip's Internet home page can be reached at: www.microchip.com.

Development Tools from Microchip	
MPLAB™	Integrated Development Environment (IDE)
MPLAB-C17	C compiler for PIC17CXXX MCUs
MPLAB-C18	C compiler for PIC18CXXX MCUs
C compiler	Sold by third-party vendors (HI-TECH, IAR, CCS)
MPASM	Universal PICmicro macro-assembler
MPLINK/MPLIB	Linker/Librarian
MPLAB-SIM	Software Simulator
MPLAB-ICD	In-circuit debugger evaluation kit
MPLAB-ICE 2000	Full-featured modular in-circuit emulator
ICEPIC	Low-cost in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART®Plus	Entry-level development kit with programmer

MPLAB™-ICE

In-Circuit Emulator



MPLAB-ICE is a high-performance, real-time in-circuit emulator.

MPLAB-ICE is Microchip's new Universal In-Circuit Emulator (ICE) for the PICmicro® 8-bit microcontrollers (MCU). Designed with the user requirements in mind, the MPLAB-ICE system is small, portable and light weight, and offers improved performance and value. For quick hook-up to portable or desktop PCs, MPLAB-ICE easily connects to the parallel (printer) port. In addition, MPLAB-ICE provides a migration path for existing customers designing with PICMASTER® Probe Kits.

Interchangeable processor modules allow the system to be easily configured to emulate different processors. This modular system consists of an emulator pod, a processor module, a device adapter, and a translation socket. Also included is Microchip's MPLAB Integrated Development Environment (IDE) featuring MPASM macro assembler, MPLAB programmer's editor, symbolic debugger, and project manager with built-in support for high-level languages that supports the Common Object Description format (i.e., MPASM and MPLAB-C17).

MPLAB-ICE 2000 is a full-featured emulator system providing full-speed emulation, low voltage operation, 32K by 128-bit trace, and up to 65,535 breakpoints. Complex triggering of the MPLAB-ICE 2000 provides sophisticated trace analysis and precision breakpoints. The trace analyzer captures real-time execution addresses, opcodes, and read/writes of external data. It also traces all file register RAM usage showing internal addresses and data values, as well as all accesses to special function registers, including I/O, timers, and peripherals. Triggers and breakpoints can be set on single events, multiple events, and sequences of events. The MPLAB-ICE 2000 analyzer is fully transparent and does not require halting the processor to view the trace. In addition, MPLAB-ICE 2000 supports code coverage profiling on program memory accesses.

Features:

- High-performance PC-based development system for PICmicro MCUs
- Includes MPLAB IDE
- Assembly and C source level debugging
- Real-time in-circuit emulation to maximum speed of PICmicro MCUs
- Program memory emulation and memory mapping capability up to 64K words
- Real-time trace with up to 32K deep by 128 bit wide buffer
- Low voltage emulation (as low as 2.0 volts)
- Unlimited software breakpoints
- Trigger/break/trace on program address and data; internal register address and data; eight external inputs; bus cycle type
- Complex breakpoints with up to four levels of advanced trigger features, including sequential events, AND/OR events, filtered trace, time between two events, and pass counts
- External trigger input and output allows logic analyzer/scope interface
- Time-stamp trace feature
- Software programmable processor clock (32 kHz to 40 MHz)
- Code coverage
- Parallel port (printer) interface
- Supports all PICmicro package types, including all through-hole and surface-mount packages
- Interchangeable processor modules

Ordering Information:

See the Development Tools Selection Chart, or www.microchip.com for specific part numbers. To order or obtain more information about MPLAB-ICE or any other Microchip product, contact the Microchip Sales Office, representative, or distributor nearest you.

Host System Requirements:

PC with 386 or higher processor. Pentium® recommended

8 MB Memory, 32 MB recommended

16 MB hard disk space, 20 MB recommended

VGA or Super VGA Monitor

Microsoft® Windows® 3.1 or greater

Parallel Port

System Description:

Features	MPLAB-ICE 2000
Real-Time Emulation	Full Speed
Low-Voltage Emulation	2.0 to 5.5 volts
Trace Memory	32K x 128 bit
Break/Trigger on Internal Registers	Yes
Software Breakpoints	Program Address
Complex Break/Trigger on Logic	Program Address and Data; Internal Register Address and Data; Access Type; and 8 External Inputs
Logic Analyzer Trigger	1 External Input and Output
Multi-level Trigger	Yes (four levels)
Pass Counter	Yes
Delay Counter	Yes
Time Stamp	Yes
Programmable Clock	32 kHz to 40 MHz
Logic Probes	Yes
Communications	Parallel (printer) Port
Code Coverage Profiling	Yes

*Contact Microchip Technology Inc for availability

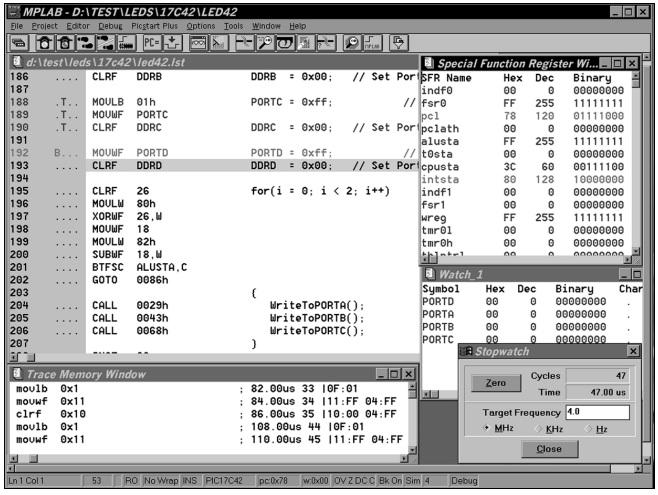
Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers, and Corporate Application Engineers. Microchip's Internet home page can be reached at: www.microchip.com.

Development Tools from Microchip	
MPLAB™	Integrated Development Environment (IDE)
MPLAB-C17	C compiler for PIC17CXXX MCUs
MPLAB-C18	C compiler for PIC18CXXX MCUs
C compiler	Sold by third-party vendors (HI-TECH, IAR, CCS)
MPASM	Universal PICmicro macro-assembler
MPLINK/MPLIB	Linker/Librarian
MPLAB-SIM	Software Simulator
MPLAB-ICD	In-circuit debugger evaluation kit
MPLAB-ICE 2000	Full-featured modular in-circuit emulator
ICEPIC	Low-cost in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART®Plus	Entry-level development kit with programmer

MPLAB™-SIM

Software Simulator



MPLAB-SIM provides an extremely powerful and cost-effective debugging solution.

The MPLAB-SIM software simulator allows speedy isolation of code problems and versatile debugging of firmware designs on all PICmicro® MCU devices. MPLAB-SIM simulates the core functions as well as many of the peripherals of the PICmicro MCU product line. External signals can be simulated with stimulus files or from user-defined key presses and can be applied to any external PICmicro MCU pin. Values can be directly injected into registers from files to simulate A/D conversions and other 8-bit inputs.

MPLAB-SIM simulates timers, counters, I/O ports, Watch Dog Timers, Capture/Compare modules, comparators, sleep mode, LCD registers, parallel slave ports, interrupts and wake up from sleep. Registers not usually accessible, such as prescalers and postscalers, can be viewed in the simulator.

MPLAB-SIM uses the same interface as the PICMASTER® and ICEPIC emulators to provide easy migration to these and other Microchip development tools.

MPLAB-SIM is a central component in the MPLAB™ Integrated Development Environment (IDE) desktop. The MPLAB IDE provides a powerful development environment supporting a rich set of software and hardware tools. This highly integrated tool set lets you develop and debug your application as a project, moving quickly from code generation, through compilation and editing, to debugging and optimization, and finally to programming PICmicro MCU devices.

Features:

MPLAB-SIM Software Simulator

- Discrete-event simulator integrated into the MPLAB Integrated Development Environment
- Simulates the core functions as well as most peripherals of the PICmicro MCUs

External Stimulus Events

- Asynchronous events defined by the user can be assigned to function keys on the keyboard.
- Synchronous clocks with adjustable duty cycles and periods can be assigned to any pin.
- Pin stimulus files can simulate multiple input signals precisely synchronized to code execution.
- Register injection files can provide 8-bit values at specific points in the program to simulate A/D conversions or byte values applied to ports or other registers.

Expanded Trace Buffer

- 8k cycles of program execution provide detailed information on program flow.
- Time stamp shows the elapsed time at each instruction execution.
- Changed register values are shown on each instruction cycle.

MPLAB Integrated Development Environment

- Project Manager to easily build applications
- Programmers editor for creating and modifying source code
- MPASM/MPLINK/MPLIB universal assembler/linker/librarian
- Support for MPLAB-C17 and third party C compilers
- Source level symbolic debugging
- Stopwatch with programmable clock frequency
- Watch windows
- PICSTART® Plus and PRO MATE® device programmer support
- Extensive on line help
- Programmable tool bars, powerful breakpointing features, and much more!

MPLAB™-SIM

Ordering Information:

Model Name:

MPLAB-SIM

Ordering Part Number:

SW007002

Tools Supported:

MPLAB-SIM Software Simulator

PICMASTER In-Circuit Emulator

PICSTART Plus Low-cost Programmer

PRO MATE II Device Programmer

ICEPIC Low-cost Emulator

Host System Requirements:

PC with 386 or higher processor. Pentium® recommended

8 MB Memory, 32 MB recommended

12 MB hard disk space, 20 MB recommended

VGA or Super VGA Monitor

Microsoft® Windows® 3.1 or greater

System Description:

MPLAB-SIM is a discrete event simulator software application designed to imitate operation of the PICmicro MCUs. It allows the user to debug software that will use any of these microcontrollers.

At any instruction boundary, you may examine and/or modify any data area within the processor, or provide external stimulus to any of the pins. MPLAB-SIM gives you a solid, low cost, source-level debug tool to help you through the early design verification stages of your project.

MPLAB-SIM runs under MPLAB. This allows you to write, debug, and optimize the PICmicro MCU applications for firmware product designs.

MPLAB IDE software package includes the following:

- MPLAB Project Manager
- MPLAB Editor
- MPASM Universal Macro Assembler for the PICmicro MCUs and other language products supporting the Common Object Description file format

To order or obtain more information about MPLAB-SIM, contact the Microchip sales office nearest you. MPLAB-SIM software may also be downloaded from Microchip's internet home page at www.microchip.com.

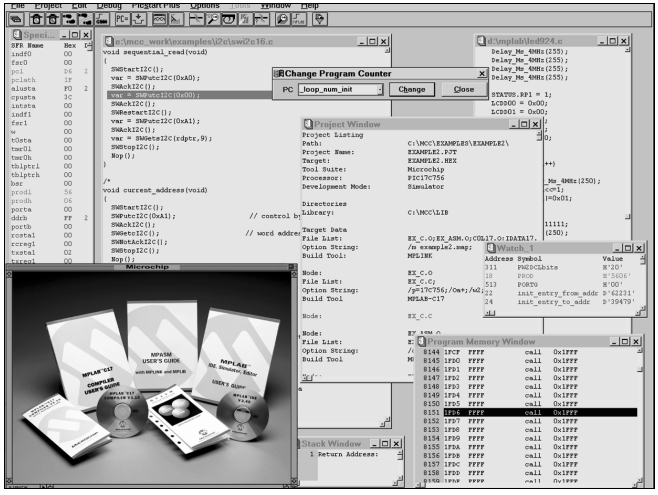
Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers, and Corporate Application Engineers. Microchip's Internet home page can be reached at: www.microchip.com.

Development Tools from Microchip	
MPLAB™	Integrated Development Environment (IDE)
MPLAB-C17	C compiler for PIC17CXXX MCUs
MPLAB-C18	C compiler for PIC18CXXX MCUs
C compiler	Sold by third-party vendors (HI-TECH, IAR, CCS)
MPASM	Universal PICmicro macro-assembler
MPLINK/MPLIB	Linker/Librarian
MPLAB-SIM	Software Simulator
MPLAB-ICD	In-circuit debugger evaluation kit
MPLAB-ICE 2000	Full-featured modular in-circuit emulator
ICEPIC	Low-cost in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART®Plus	Entry-level development kit with programmer

MPLAB™-C17

ANSI-Compliant C Compiler for PIC17CXXX Microcontrollers



MPLAB-C17 provides powerful integration capabilities and ease of use!

The MPLAB-C17 compiler is a full-featured ANSI compliant C compiler for the Microchip Technology PIC17CXXX family of PICmicro® 8-bit microcontrollers (MCU). MPLAB-C17 is fully compatible with Microchip's MPLAB Integrated Development Environment (IDE), allowing source level debugging with both the MPLAB-ICE In-Circuit Emulator and the MPLAB-SIM simulator. MPLAB provides a convenient, project oriented development environment that reduces development time.

MPLAB-C17 allows code for the PIC17CXXX family to be written in the C high-level language using powerful PICmicro libraries, enabling the developer to devote more time to the application and less time to the details of the processor.

MPLAB-C17 was designed explicitly for the PIC17CXXX family and allows the use of a software stack for maximum RAM reusability or can be run without a stack for optimal code space efficiency.

MPLAB-C17 provides user configurable interrupt support macros for saving and restoring context during interrupt handling. Libraries and interrupt handlers are provided for multiple memory models. Libraries, precompiled objects and linker scripts can be included in MPLAB projects along with C and Assembly source files for use with MPLAB's make and build functions.

MPLAB-C17 will run on any 386 or better PC, on DOS® 5.0+ or as a native 32-bit Windows® 95 or Windows NT® executable.

Features:

- ANSI compliant
- Integrated with MPLAB for easy-to-use project management and source-level debugging
- Generates relocatable object modules for enhanced code reuse
- Fully compatible with object modules generated with MPASM, allowing complete freedom in mixing assembly and C in a single project
- Transparent read/write access to external memory
- Interrupt code can be written in C or Assembly
- Strong support for inline assembly for when total control is absolutely necessary
- Efficient code generator engine with multi-level optimization
- Extensive library support, including PWM, SPI™, I2C™, USART, UART, string manipulation, and math libraries
- Allows code and data to be located at absolute addresses
- Easy manipulation of processor configuration words

MPLAB™-C17

Ordering Information:

Model Name:

MPLAB-C17

Ordering Part Number:

SW006010

Devices Supported:

All PIC17CXXX microcontrollers

Host System Requirements:

PC with 386 or higher processor

4 MB Memory, 16 MB recommended

8 MB hard disk space,
20 MB recommended

VGA or Super VGA Monitor

MS-DOS/PC-DOS
version 5.0 or greater
or Microsoft® WIndows 95 or
WIndows NT

System Description:

The MPLAB-C17 ANSI-compliant C Compiler comes complete with the MPLAB IDE. The IDE allows you to quickly move between different development and debugging modes, for example, you can quickly advance from software debugging with MPLAB-SIM to hardware debugging with MPLAB-ICE.

MPLAB-C17 has implemented extensions to the C language to provide specific support for Microchip's PICmicro MCU environment. These C library extensions include:

A/D converter	Input Capture	Interrupt Support Macros
SPI	Timers	USART
I^2C	I/O Port	Pulse Width Modulation
Reset	External LCD	Software SPI
Software I^2C	Software USART	Character Classification
Relay	Memory/String Manipulation	Number/Text Conversion
32-bit Math Library		

To order or obtain more information about MPLAB-C17 or any other Microchip product, contact the Microchip sales office nearest you.

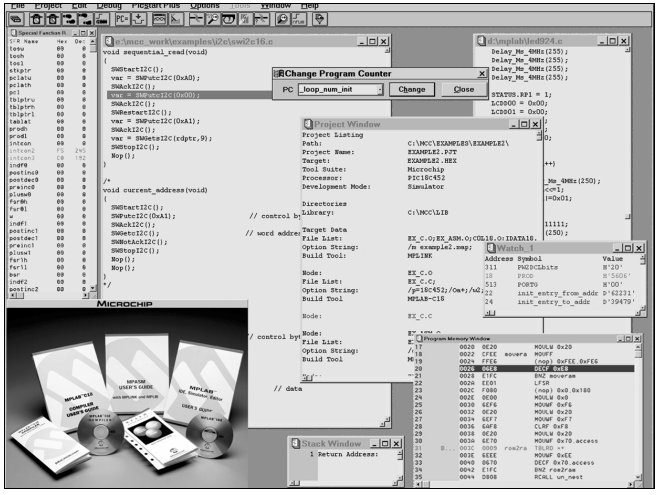
Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers, and Corporate Application Engineers. Microchip's Internet home page can be reached at: www.microchip.com.

Development Tools from Microchip	
MPLAB™	Integrated Development Environment (IDE)
MPLAB-C17	C compiler for PIC17CXXX MCUs
MPLAB-C18	C compiler for PIC18CXXX MCUs
C compiler	Sold by third-party vendors (HI-TECH, IAR, CCS)
MPASM	Universal PICmicro macro-assembler
MPLINK/MPLIB	Linker/Librarian
MPLAB-SIM	Software Simulator
MPLAB-ICD	In-circuit debugger evaluation kit
MPLAB-ICE 2000	Full-featured modular in-circuit emulator
ICEPIC	Low-cost in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART®Plus	Entry-level development kit with programmer

MPLAB™-C18

ANSI-Compliant C Compiler for PIC18CXXX Microcontrollers



MPLAB-C18 provides powerful integration capabilities and ease of use!

The MPLAB-C18 compiler is a full-featured ANSI compliant C compiler for the Microchip Technology PIC18CXXX family of PICmicro® 8-bit microcontrollers (MCUs). MPLAB-C18 is fully compatible with Microchip's MPLAB Integrated Development Environment (IDE), allowing source level debugging with both the MPLAB-ICE In-Circuit Emulator and the MPLAB-SIM simulator. MPLAB provides a convenient, project oriented development environment that reduces development time.

MPLAB-C18 allows code for the PIC18CXXX family to be written in the C high-level language using powerful PICmicro libraries, enabling the developer to devote more time to the application and less time to the details of the processor.

MPLAB-C18 was designed explicitly for the PIC18CXXX family and allows the use of a software stack for maximum RAM reusability.

MPLAB-C18 provides user configurable interrupt support for saving and restoring context during interrupt handling. Libraries are provided for multiple memory models. Libraries, precompiled objects, and linker scripts can be included in MPLAB projects along with C and Assembly source files for use with MPLAB's make and build functions.

MPLAB-C18 will run on any 486 or better PC as a native 32-bit Windows®95 or Windows NT®executable.

Features:

- ANSI compliant
- Integrated with MPLAB for easy-to-use project management and source-level debugging
- Generates relocatable object modules for enhanced code reuse
- Fully compatible with object modules generated with MPASM, allowing complete freedom in mixing C and Assembly in a single project
- Transparent read/write access to external memory
- Interrupt code can be written in C or Assembly
- Strong support for inline assembly for when total control is absolutely necessary
- Efficient code generator engine with multi-level optimization
- Extensive library support, including peripheral string manipulation, and math libraries
- Allows code and data to be located at absolute addresses
- Easy manipulation of processor configuration words

MPLAB™-C18

Ordering Information:

Model Name:

MPLAB-C18

Ordering Part Number:

SW006011

Devices Supported:

All PIC18CXXX microcontrollers

Host System Requirements:

PC with 486 or higher processor

4 MB Memory, 16 MB recommended
8 MB hard disk space,
20 MB recommended

MS-DOS®/PC-DOS® or
Microsoft® Windows® 95 or
Windows NT®

System Description:

The MPLAB-C18 ANSI-compliant C Compiler comes complete with the MPLAB IDE. The IDE allows you to quickly move between different development and debugging modes, for example, you can quickly advance from software debugging with MPLAB-SIM to hardware debugging with MPLAB-ICE.

MPLAB-C18 has implemented extensions to the C language to provide specific support for Microchip's PICmicro MCU environment.

To order or obtain more information about MPLAB-C18 or any other Microchip product, contact the Microchip sales office nearest you.

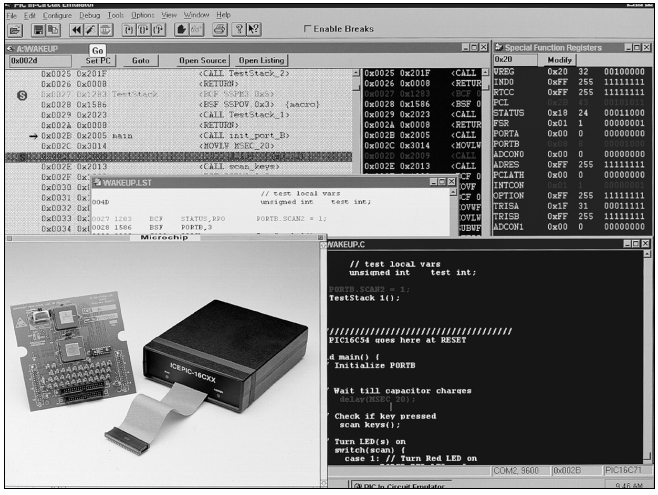
Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers, and Corporate Application Engineers. Microchip's Internet home page can be reached at: www.microchip.com

Development Tools from Microchip	
MPLAB™	Integrated Development Environment (IDE)
MPLAB-C17	C compiler for PIC17CXXX MCUs
MPLAB-C18	C compiler for PIC18CXXX MCUs
C compiler	Sold by third-party vendors (HI-TECH, IAR, CCS)
MPASM	Universal PICmicro macro-assembler
MPLINK/MPLIB	Linker/Librarian
MPLAB-SIM	Software Simulator
MPLAB-ICD	In-circuit debugger evaluation kit
MPLAB-ICE 2000	Full-featured modular in-circuit emulator
ICEPIC	Low-cost in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART®Plus	Entry-level development kit with programmer

ICEPIC

Low-Cost PIC16CXXX In-Circuit Emulator



ICEPIC: Affordable PIC16CXXX In-Circuit Emulation Solution.

ICEPIC is a low-cost in-circuit emulation solution for the Microchip Technology PIC16C5X and PIC16CXXX families of 8-bit one-time-programmable (OTP) microcontrollers. The modular system can support different subsets of PIC16C5X or PIC16CXXX products through the use of interchangeable personality modules or daughter boards. The emulator is capable of emulating without target application circuitry being present.

ICEPIC is designed to operate on PC-compatible machines ranging from 286-AT® systems through the new Pentium® based machines. The ICEPIC development software runs under Microsoft® Windows® 3.X environment, allowing the operator access to a wide range of supporting software accessories.

The ICEPIC development software provides a user-friendly operating environment with an easy-to-use toolbar; unlimited number of breakpoints; single, multiple and procedure step; ability to display and modify any register; user-selectable processor speeds via an oscillator module; full context-sensitive help and an RS-232 serial port.

ICEPIC is fully compatible with Microchip's MPASM Universal Assembler and MPLAB-C17 Compiler.

Features:

- Real time, non-intrusive emulation of PIC16C5X and PIC16CXXX microcontrollers
- 8K words of emulation memory
- Full speed, real time emulation to 20 MHz for PIC16C5X family
- Up to 10 MHz real time emulation for PIC16CXXX family
- Microsoft Windows compatible
- Source level debug capability in assembly or C
- Symbolic debug capability
- 8K hardware breakpoints
- Custom watch points
- PC communication via serial interface at speeds up to 57K baud
- Display and modify any register (Program or Data)
- User selectable processor speeds (via oscillator module)

Ordering Information:

See the Development Tools Selection Chart, page 8-1 or www.microchip.com for specific part numbers. To order or obtain more information about MPLAB-ICE or any other Microchip product, contact the Microchip Sales Office, representative, or distributor nearest you.

System Description:

The low-cost PC-based ICEPIC In-Circuit Emulator system comes with an emulator unit (mother board), power supply, RS-232 cable, probe header cable(s) to connect to the application circuit, and one device-specific personality daughter board.

These interchangeable personality modules or daughter boards are contained with the mother board within one housing, connecting to the target application via a connector cable that extends from the housing. The mother board incorporates the common emulation logic while the daughter board is for device-specific emulator logic. This economical system allows the user to purchase a new daughter board for a new processor group as needed, at approximately 30% of the full system cost.

ICEPIC was designed by NEOSOFT Inc. and is manufactured under license by RF Solutions Ltd. To order or obtain more information about ICEPIC or any other Microchip product, contact the Microchip sales office nearest you.

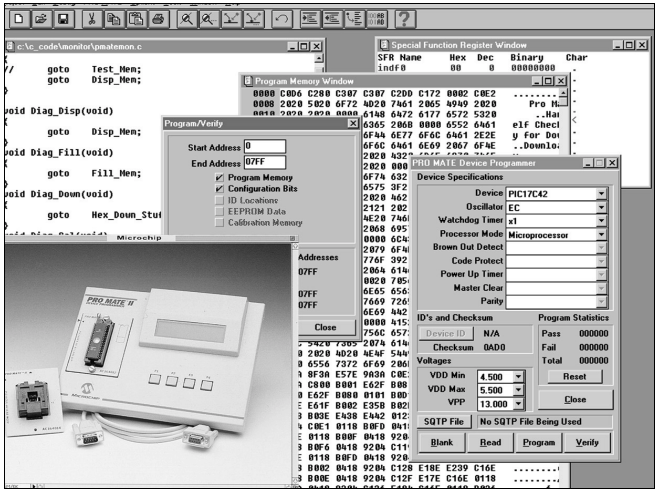
Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers, and Corporate Application Engineers. Microchip's Internet home page can be reached at: www.microchip.com.

Development Tools from Microchip	
MPLAB™	Integrated Development Environment (IDE)
MPLAB-C17	C compiler for PIC17CXXX MCUs
MPLAB-C18	C compiler for PIC18CXXX MCUs
C compiler	Sold by third-party vendors (HI-TECH, IAR, CCS)
MPASM	Universal PICmicro macro-assembler
MPLINK/MPLIB	Linker/Librarian
MPLAB-SIM	Software Simulator
MPLAB-ICD	In-circuit debugger evaluation kit
MPLAB-ICE 2000	Full-featured modular in-circuit emulator
ICEPIC	Low-cost in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART®Plus	Entry-level development kit with programmer

PRO MATE® II

Universal Microchip Device Programmer



CE Compliant PRO MATE II makes programming easy.

The PRO MATE II device programmer tool allows development engineers to program user software into Microchip Technology's entire line of PICmicro® 8-bit microcontrollers (MCU), HCS Security Products, and 2- and 3-wire serial EEPROM products.

The PRO MATE II device programmer is easy to use and operates either as a stand-alone unit or in conjunction with a PC-compatible host system. When connected to a host system, PRO MATE II provides an exceptionally user-friendly interface to give the developer complete control over the programming session. This time-saving tool comes complete with all the accessories needed to connect to a host system including interface cables and a universal input power supply.

In addition to the programmer unit, the PRO MATE II system contains Microchip's highly-acclaimed MPLAB™ Integrated Development Environment (IDE), with its built-in editor, assembler, and Windows®-based MPLAB-SIM simulator. The

PRO MATE II programmer includes full documentation and software.

PRO MATE II is CE compliant, meaning it meets or exceeds all the directives for safety, emissions, electrostatic discharge (ESD) and susceptibility (to radiated emission) requirements set forth by the European Union (EU) countries.

The PRO MATE II device programmer is designed to be robust and reliable with: enhanced socket module alignment with four auto alignment pins; three levels of over-current protection and superior ESD immunity for rugged environments; a small and compact universal IEC power supply and improved LCD display and buttons.

Features:

- Programs EPROM and/or EEPROM program and data memory for all Microchip PICmicro MCUs, HCS Security Products, and 2- and 3-wire serial EEPROM products
- Designed to operate with the PRO MATE II In-Circuit Serial Programming™ Kit (sold separately)
- Three operating modes: Host Mode, Safe Mode, and Stand-Alone Mode
- Complete line of interchangeable socket modules supports all package options (sold separately)
- Universal platform can quickly and easily support future Microchip products
- SQTPSM serialization adds a unique serial number to each device programmed (in PC host mode)
- MPLAB Project support to automatically download object file to PRO MATE II
- MPASM Assembler translates assembler source code to object code for all PICmicro devices
- MPLAB-SIM Windows-based simulator designed to model operation of all PICmicro MCUs
- Indexed on-line help
- Complete documentation including all User's Guides and Microchip's Technical Library CD-ROM
- Supports the serial programming mode in PICmicros
- Supports a DOS command line interface

PRO MATE® II

Ordering Information:

Model Name:

PRO MATE II

Ordering Part Number:

DV007003

Devices Supported:

All PICmicro MCUs

HCS Security Products

Serial EEPROMs

Socket Modules and ICSP Kit are sold separately. For a complete list, please contact Microchip or refer to the Development Systems Ordering Guide (DS30177).

Host System Requirements:

PC with 386 or higher processor. Pentium® recommended

8 MB Memory, 32 MB recommended

16 MB hard disk space, 20 MB recommended

VGA or Super VGA Monitor

Microsoft® Windows 3.1 or greater

CD-ROM Drive

COM Port

System Description:

Microchip's PRO MATE II makes it easy to program the company's entire line of 8-bit RISC microcontrollers, HCS Security Products, and serial EEPROMs operating either as a stand-alone unit or in conjunction with a PC-compatible host system. When connected to a host system, PRO MATE II provides an exceptionally user-friendly interface to give the developer complete control over the programming session.

The new PRO MATE II software provides many user interface features. These include a "safe mode," where accidental corruption of master code is prevented, and the ability to save and restore "environment" settings. The PRO MATE II system runs with Microchip's popular MPLAB IDE software.

To order or obtain more information about PRO MATE II or any other Microchip product, contact the Microchip sales office nearest you.

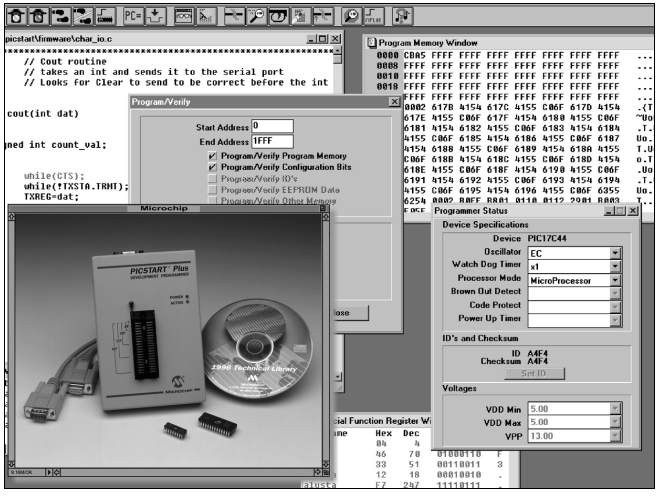
Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers, and Corporate Application Engineers. Microchip's Internet home page can be reached at: www.microchip.com.

Development Tools from Microchip	
MPLAB™	Integrated Development Environment (IDE)
MPLAB-C17	C compiler for PIC17CXXX MCUs
MPLAB-C18	C compiler for PIC18CXXX MCUs
C compiler	Sold by third-party vendors (HI-TECH, IAR, CCS)
MPASM	Universal PICmicro macro-assembler
MPLINK/MPLIB	Linker/Librarian
MPLAB-SIM	Software Simulator
MPLAB-ICD	In-circuit debugger evaluation kit
MPLAB-ICE 2000	Full-featured modular in-circuit emulator
ICEPIC	Low-cost in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART®Plus	Entry-level development kit with programmer

PICSTART® Plus

Low-cost Development Kit Supports All PICmicro® MCUs



PICSTART Plus makes designing with Microchip MCUs simple and affordable.

The PICSTART Plus development system from Microchip Technology provides the product development engineer with a highly-flexible, low-cost microcontroller (MCU) design tool set for all Microchip PICmicro 8-bit devices (DIP packages up to 40 pins).

The PICSTART Plus development system runs on any PC-compatible machine running under the Windows® 3.1 operating system. PICSTART Plus is easy-to-use and features Microchip's highly acclaimed MPLAB™ Integrated Development Environment (IDE), with its built-in editor, assembler and Windows-based

MPLAB-SIM simulator. The PICSTART Plus development system includes full documentation, software, development programmer, and a device sample.

The CE compliant PICSTART Plus development programmer features a molded plastic enclosure and special circuit design techniques to enhance ESD protection. PICSTART Plus is a development programmer and is not recommended for use in a production environment.

Sample software programs are provided to help the developer quickly become familiar with the PICSTART Plus development system and with Microchip's PICmicro MCU families. The PICSTART Plus system also includes Microchip's new CD-ROM containing complete documentation necessary to get started with your design.

Features:

- Operates with PC-compatible host system running Windows under MPLAB environment
- Reads, programs, verifies EPROM and EEPROM program and data memory
- Reads, programs, verifies all configuration bits
- Programs and verifies an address range
- Displays, edits, and transfers device contents to and from programmer unit
- MPLAB Project support to automatically download object file to PICSTART Plus
- MPASM Assembler translates assembler source code to object code for all PICmicro devices
- MPLAB-SIM Windows-based simulator designed to model operation of all PICmicro MCUs
- Complete with RS-232 cable and 9 volt power supply
- PICmicro MCU device sample
- Complete documentation, User's Guides and CD-ROM

PICSTART® Plus

Ordering Information:

Model Name:

PICSTART Plus

Ordering Information:

DV003001

Devices Supported:

All PICmicro MCUs
(DIP Packages up to 40 pins)

Host System Requirements:

PC with 386 or higher processor.
Pentium® recommended

8 MB Memory, 32 MB recommended

16 MB hard disk space,
20 MB recommended

VGA or Super VGA Monitor

Microsoft® Windows 3.1 or greater

CD-ROM Drive

COM Port

System Description:

The PICSTART Plus development system includes the PICSTART Plus development programmer and the MPLAB IDE.

The PICSTART Plus programmer gives the product developer the ability to program user software into any of the supported MCUs The PICSTART Plus software running under MPLAB provides for full interactive control over the programmer.

The MPASM macro assembler provides programmable memory data files, listing files, and special files required for symbolic debug. The MPLAB-SIM software simulator allows the user to isolate code problems and debug firmware designs on PICmicro devices. It simulates the core functions as well as most of the peripherals of the PICmicro MCU families. It is particularly suitable for optimizing algorithms where real-time emulation is not required.

To order or obtain more information about PICSTART Plus or any other Microchip product, contact the Microchip sales office nearest you.

Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers, and Corporate Application Engineers. Microchip's Internet home page can be reached at: www.microchip.com

Development Tools from Microchip	
MPLAB™	Integrated Development Environment (IDE)
MPLAB-C17	C compiler for PIC17CXXX MCUs
MPLAB-C18	C compiler for PIC18CXXX MCUs
C compiler	Sold by third-party vendors (HI-TECH, IAR, CCS)
MPASM	Universal PICmicro macro-assembler
MPLINK/MPLIB	Linker/Librarian
MPLAB-SIM	Software Simulator
MPLAB-ICD	In-circuit debugger evaluation kit
MPLAB-ICE 2000	Full-featured modular in-circuit emulator
ICEPIC	Low-cost in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART®Plus	Entry-level development kit with programmer

MICROCHIP

KEELOQ®

Evaluation Kit

Features:

- Transmit hopping code messages using KeeLoq encoders
- Display the fixed and changing parts of hopping code messages on a PC
- Receive and decode hopping code transmissions using a PIC16C56/ 93C46 decoder or HCS512 decoder
- Learn new encoders onto the HCS512 decoders or PIC16C56/ 93C46 EEPROM-based decoder
- Program KeeLoq encoders
- Comprehensive User's Guide
- Schematics for transmitter and receiver
- Supports HCS200, HCS300, HCS301, HCS360, and HCS361 encoders
- Licence disk containing source code and application notes describing a software decoder.

Lets you evaluate the capabilities of Microchip's code hopping devices.

The KEELOQ Evaluation Kit demonstrates the capabilities of Microchip Technology's code hopping technology. The KEELOQ Code Hopping Encoder devices are designed to be the transmitter portion. The KeeLoq Decoder devices are used in the receiver portion of secure RKE* systems. The devices use the KeeLoq code hopping algorithm which combines high security, a small package outline, and a very low cost to make this an ideal solution for unidirectional RKE systems.

Primary applications for the KeeLoq devices include automobile keyless entry and security systems, garage door openers, home security systems, central locking systems, gate openers, vehicle immobilizers, identity tokens, and a growing list of other applications.

The KEELOQ Evaluation Kit includes all the necessary hardware to evaluate a code hopping system, as well as a basic demonstration software program, which is supplied on a PC-compatible 3.5-inch diskette.

*Remote Keyless Entry

Ordering Information:

Model Name:

KEELOQ Evaluation Kit

Ordering Part Number:

DM303002

Devices Supported:

HCS200 Encoder

HCS300 Encoder

HCS301 Encoder

HCS360 Encoder

HCS361 Encoder

HCS512 Decoder

PIC16C56 Decoder

Host System Requirements:

PC-compatible computer: 386DX, 486 or Pentium®-based with ISA or EISA bus

Serial Port

EGA, VGA, 8514/A, Hercules graphic card (EGA or higher recommended)

Microsoft® Windows® 3.1 or greater in 386 enhanced mode

4 MB RAM, 4 MB free disk space

3.5-inch diskette drive

System Description:

- HCS512 and PIC16C56/93C46 EEPROM-based decoders
- RF Receiver module
- On-board +5V regulator and filtered rectifier for direct input from 9V AC/DC wall adapter
- Tow radio frequency transmitters using HCS300 hopping code encoders
- RS-232 socket and associated hardware for direct connection to RS-232 interface
- Six bright LEDs connected to decoder outputs
- Supports HCS200, HCS300, HCS301, HCS360, and HCS361 samples

Also includes all necessary software to program transmitters and to display codes or messages.

To order or obtain more information about KeeLoq or any other Microchip product, contact the Microchip sales office nearest you.

Customer Support:

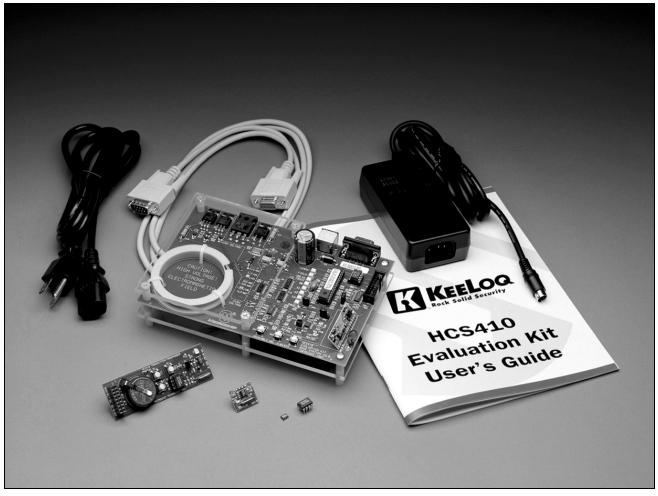
Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers, and Corporate Application Engineers. Microchip's Internet home page can be reached at: www.microchip.com

Development Tools from Microchip	
MPLAB™	Integrated Development Environment (IDE)
MPLAB-C17	C compiler for PIC17CXXX MCUs
MPLAB-C18	C compiler for PIC18CXXX MCUs
C compiler	Sold by third-party vendors (HI-TECH, IAR, CCS)
MPASM	Universal PICmicro macro-assembler
MPLINK/MPLIB	Linker/Librarian
MPLAB-SIM	Software Simulator
MPLAB-ICD	In-circuit debugger evaluation kit
MPLAB-ICE 2000	Full-featured modular in-circuit emulator
ICEPIC	Low-cost in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART®Plus	Entry-level development kit with programmer

MICROCHIP

KEELOQ®

Transponder Evaluation Kit



Lets you evaluate the capabilities of Microchip's Code Hopping transmitter and transponder devices.

The KEELOQ Transponder Evaluation Kit allows the user to fully evaluate the HCS410 and HCS412 Code Hopping Transmitter/Transponder. The HCS410 and HCS412 uses the KEELOQ code hopping technology which combines high security transmitter and transponder operation in a small, low-cost package.

Primary applications for the KeeLoq devices include automobile keyless entry, security systems, garage door openers, home security systems, central locking systems, gate openers, vehicle immobilizers, identity tokens, electronic tagging, passive (batteryless) transponders, and a growing list of other applications.

The KEELOQ Transponder Evaluation Kit is supplied with software to let the user easily reprogram or modify the HCS410 or HCS412's settings. The hardware includes a base station which functions as an HCS410/412 programmer/code hopping decoder and transponder reader unit.

Features:

- Base station with RF and inductive communication
- Allows full testing of the HCS410/412 in all its modes
- Allows programming of the HCS410/412
- Base station learns up to four transmitter/transponders
- Simple, normal, and secure learning schemes available
- Displays IFF challenge, responses, and decrypted responses
- Displays code hopping transmissions
- Windows® software includes
  - Selection of manufacturer's code
  - Selection of other key generation options
  - Selection of HCS410/412 options
  - HCS410/412 programming
  - Monitoring of IFF and code hopping transmissions
- Comprehensive User's Guide

Ordering Information:

Model Name:

KEELOQ Transponder Evaluation Kit

Ordering Information:

DM303005

Devices Supported:

HCS410 Encoder

HCS412 Encoder

Host System Requirements:

PC-compatible computer: 386DX, 486 or Pentium®-based with ISA or EISA bus

EGA, VGA, 8514/A, Hercules graphic card (EGA or higher recommended)

Microsoft® Windows® 3.1 or greater in 386 enhanced mode

4 MB RAM, 4 MB free disk space

3.5-inch diskette drive

One serial port (2400 baud)

System Description:

The KEELOQ Transponder Evaluation Kit Hardware consists of a base station, a transmitter/transponder, a batteryless transponder, and various HCS410 and HCS412 samples. The base station doubles as a programmer and decoder. The base station includes a coil used for generating a magnetic field used to communicate with a transponder inductively. The base station has an RF receiver for receiving KeeLoq code hopping transmissions.

The accompanying Windows software is supplied on a 3.5-inch diskette and includes all the necessary software for programming and testing the HCS410 and HCS412 in all its modes.

To order or obtain more information about KeeLoq or any other Microchip product, contact the Microchip sales office nearest you.

Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers, and Corporate Application Engineers. Microchip's Internet home page can be reached at: www.microchip.com

Development Tools from Microchip	
MPLAB™	Integrated Development Environment (IDE)
MPLAB-C17	C compiler for PIC17CXXX MCUs
MPLAB-C18	C compiler for PIC18CXXX MCUs
C compiler	Sold by third-party vendors (HI-TECH, IAR, CCS)
MPASM	Universal PICmicro macro-assembler
MPLINK/MPLIB	Linker/Librarian
MPLAB-SIM	Software Simulator
MPLAB-ICD	In-circuit debugger evaluation kit
MPLAB-ICE 2000	Full-featured modular in-circuit emulator
ICEPIC	Low-cost in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART®Plus	Entry-level development kit with programmer

PICDEM-1

Low-Cost PICmicro® Demonstration Board

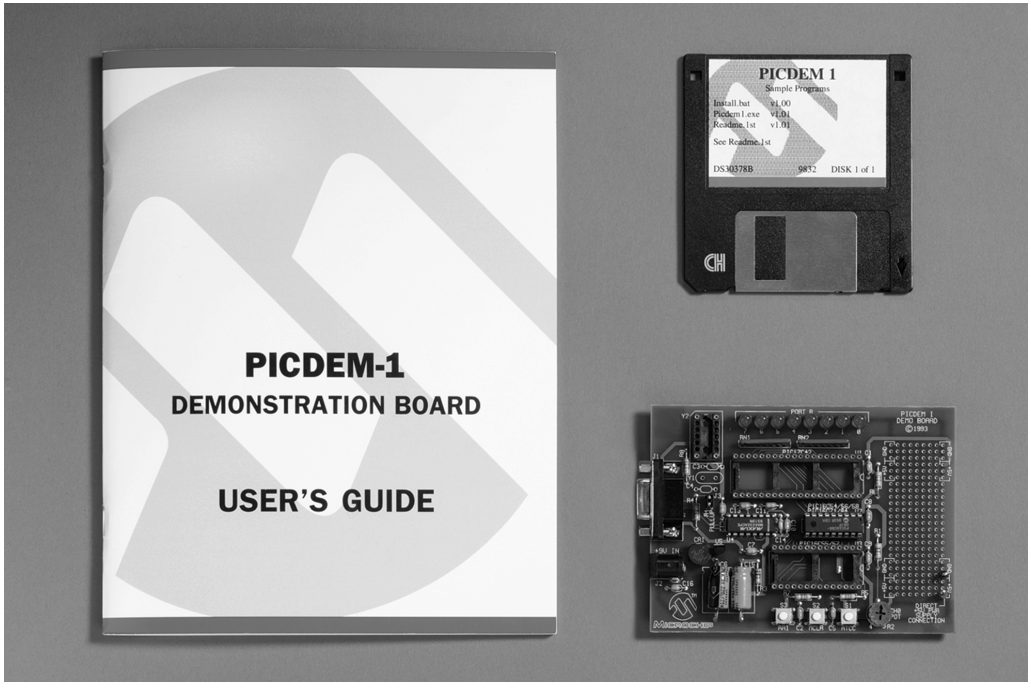
PRODUCT INFORMATION

The PICDEM-1 is a simple board which demonstrates the capabilities of several Microchip microcontrollers. The microcontrollers supported are: PIC16C5X (PIC16C54 to PIC16C58), PIC16C62X, PIC16CE62X, PIC16C71, PIC16C84, PIC17C42, PIC17C43 and PIC17C44, PIC16F84, PIC16C710, PIC16C711, PIC16C770, PIC16C771, PIC16C55X, and PIC16C715. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5-inch disk. The users can program the samples (one each of PIC17C42, PIC16C71 and PIC16C55) provided with the PICDEM-1, on a PRO MATE®or PICSTART® programmer and easily debug/test the sample code, or the user can connect the PICDEM-1 with the PICMASTER™ emulator and download the sample code to the emulator and debug/test the code. Additionally, a generous 200-hole prototype area is available for the user to build some additional hardware and connect it to the microcontroller socket(s).

FEATURES:

Hardware:

- 40-pin, 28-pin and 18-pin Precision sockets for all supported microcontrollers.
- On board +5V regulator and filter rectifier for direct input from 9V AC/DC wall adapter.
- RS-232 socket and associated hardware for direct connection to RS-232 interface.
- 5K pot to simulate analog input for PIC16C71.
- Three push button Key for external stimulus and RESET.
- Eight bright LEDs connect to PORTB, help in
- displaying 8-bit binary values on PORTB.
- Socket for “canned” crystal Oscillator.
- Unpopulated holes provided for Xtal connection
- Jumper to disconnect on board RC Oscillator.
- 200-hole prototype area for user’s hardware.



Software:

- Program for PIC16C71 to demonstrate on-chip A/D features.
- Program for PIC16C84 to demonstrate on-chip EEPROM.
- Program for PIC17C42 to demonstrate on-chip USART.
- Program for PIC16C5X to demonstrate key input capability.
- All demo programs supplied on 3.5" disk,

DOCUMENTATION

- A comprehensive User's Guide with easy to follow step-by-step Getting Started and a Tutorial.
- Schematics for the entire circuit.

SAMPLES

Several UV erasable devices supplied are included. The device types may change from time to time. The supplied devices are typically:

- PIC17C42
- PIC16C71
- PIC16C55

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
DM163001	Low-cost Demonstration Board for PIC16C5X (PIC16C54 to PIC16C58), PIC16C62X, PIC16CE62X, PIC16C71, PIC16C84, PIC17C42, PIC17C43 and PIC17C44, PIC16F84, PIC16C710, PIC16C711, PIC16C770, PIC16C771, PIC16C55X, and PIC16C715

PICDEM-2

Low-Cost PIC16CXX Demonstration Board

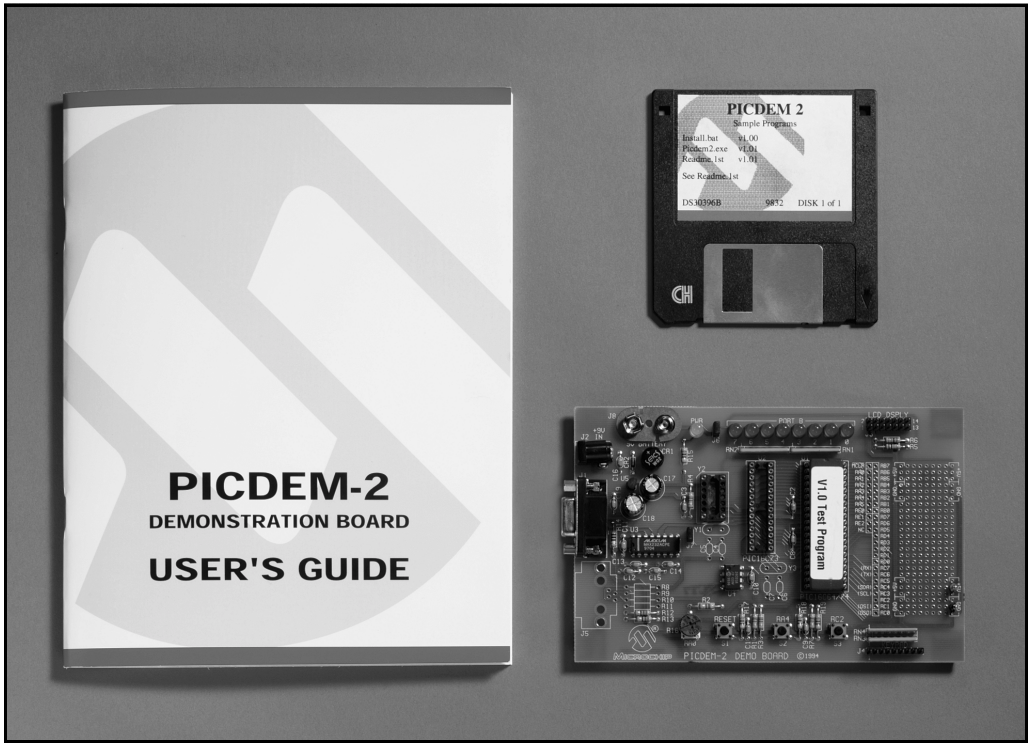
PRODUCT INFORMATION

The PICDEM-2 is a simple board which demonstrates the capabilities of several Microchip microcontrollers, including PIC16C62, PIC16C63, PIC16C64, PIC16C65, PIC16C66, PIC16C67, PIC16C72 PIC16C73, PIC16C74, PIC16C76, PIC16C77, PIC16C662, PIC16C642, PIC16F87X, PIC16C773, PIC16C774, and PIC18CXX2. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5" disk. A programmed sample is included, and the user may erase it and program it with the other sample programs using the PRO MATE® or PICSTART® Plus programmer and easily debug and test the sample code. The PICDEM-2 is also usable with the PICMASTER™ emulator, and all of the sample programs can be run and modified using the PICMASTER. Additionally, a generous prototype area is available for user hardware.

FEATURES:

Hardware:

- 40- and 28-pin DIP sockets
- On board +5V regulator for direct input from 9V AC/DC wall adapter or 9V battery.
- RS-232C socket and associated hardware for direct connection to RS-232C interface.
- 5K pot for analog inputs for the PIC16C73/74
- Three push button keys for external stimulus and RESET.
- Eight bright LEDs connected to PORTB for displaying 8-bit binary values.
- Socket for “canned” crystal oscillator.
- Unpopulated holes provided for crystal connection
- 128 x 8 Serial EEPROM.
- LCD module header.
- Keyboard header.
- Unpopulated holes for ACCESS.bus™ connector.



Hardware (continued):

- Jacks for connection of 9V battery.
- Jumper to disconnect on-board RC oscillator.
- Prototype area for user hardware.

Software:

- Program for PIC16C74 to demonstrate on-chip A/D feature.
- Program for PIC16C64 to demonstrate I^2C™ Serial EEPROM usage.
- All demo programs supplied on 3.5-inch disk.

DOCUMENTATION:

- A comprehensive User's Guide with easy to follow, step-by-step Getting Started and Tutorial.
- Full schematics.

Samples:

Several UV erasable devices supplied are included. The device types may change from time to time. The supplied devices are typically:

- PIC16C64
- PIC16C74

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
DM163002	Low-cost Demonstration Board for PIC16C62, PIC16C63, PIC16C64, PIC16C65, PIC16C66, PIC16C67, PIC16C72 PIC16C73, PIC16C74, PIC16C76, PIC16C77, PIC16C662, PIC16C642, PIC16F87X, PIC16C773, PIC16C774, and PIC18CXX2

PICDEM-3

Low-Cost PIC16C9XX Demonstration Board

PRODUCT INFORMATION

The PICDEM-3 is a simple board which demonstrates the capabilities of Microchip's microcontroller LCD family. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5" disk. A programmed sample is included, and the user may erase it and program it with the other sample programs using the PRO MATE® or PICSTART®Plus programmer. The PICSTART Plus requires a 68- to 40-pin converter socket.

The PICDEM-3 is also usable with the PICMASTER® emulator, and all of the sample programs can be run and modified using the PICMASTER. Additionally, a generous prototype area is available for user hardware.

All software is written in C using the MPLAB-C17/demo version.

FEATURES:

Hardware:

- 68-pin PLCC sockets
- Unpopulated 44-pin PLCC socket for future 44-pin versions
- On board +5V regulator for direct input from 9V AC/DC wall adapter or 9V battery
- RS-232C socket and associated hardware for direct connection to RS-232C interface
- 5K pot for analog input to the PIC16C9XX
- Thermistor for use with A/D converter
- Three push button keys for external stimulus and RESET
- LCD Panel with 4 backplanes and 12 segments
- On-board LCD charge pump circuit with "unpopulated" external resistor ladder supported
- Socket for "canned" crystal oscillator
- Unpopulated holes provided for crystal or ceramic resonator connection
- Supports custom LCD Panel connection (up to 4 common, up to 28 segment) with common/segment header
- LCD pixel data converted to a digital result. Requires external hardware with SPI interface
- Serial port to communicate LCD pixel data to PC for display
- Keyboard header
- Jacks for connection of 9V battery
- Jumper to disconnect on-board RC oscillator
- Prototype area for user hardware

Software:

- MPLAB-C demo for PIC16CXX
- Sample programs which demonstrate:
  - Real-time clock and display
  - Temperature sensor and display
  - Volt meter
  - USART using the SPI port
  - PC software to display LCD pixel information
- All demo programs supplied on 3.5" disk.

DOCUMENTATION:

- A comprehensive User's Guide with easy to follow, step-by-step Getting Started and Tutorial.
- Full schematics.

A UV erasable device is supplied. The device type may vary depending upon availability. The supplied devices are either:

- PIC16C924
- PIC16C923

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
DM163003	Low-cost Demonstration Board for PIC16C923 and PIC16C924

NOTES:

PICDEM-17

PICmicro® Demonstration Board

PRODUCT INFORMATION

The PICDEM-17 is an evaluation board that demonstrates the capabilities of several Microchip microcontrollers, including PIC17C752, PIC17C756, PIC17C762, and PIC17C766. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5-inch disk. A programmed sample is included, and the user may erase it and program it with the other sample programs using the PRO MATE II or PICSTART Plus device programmers and easily debug and test the sample code. In addition, PICDEM-17 supports down-loading of programs to and executing out of external FLASH memory on board. The PICDEM-17 is also usable with the MPLAB™-ICE or PICMASTER® emulator, and all of the sample programs can be run and modified using either emulator. Additionally, a generous prototype area is available for user hardware.

FEATURES:

Hardware

- 68-pin PLCC socket
- Space for 84-pin socket
- On board digital and analog +5V regulator for direct input from 9V AC/DC wall adapter or 9V battery
- RS232 interface with DB9 connector for USART1
- RS232 interface with DB9 connector for USART2 with hardware handshaking signals
- Precision analog voltage source and reference for 10-bit A/D
- MCP2510 CAN interface support
- External memory mapped 64K x 16 bit FLASH
- 24LC01B Serial EEPROM
- 8 memory mapped push button switches
- 8 memory mapped LEDs
- "Canned" oscillator for PICmicro and CAN chip
- LCD module header
- Analog and digital prototyping areas

Software

- Program for PIC17C756A to demonstrate peripherals on-chip or demo board itself:
- A/D
- Capture
- FLASH
- I2C
- PWM
- Switches
- USART2
- External LCD

Documentation:

- A comprehensive User's Guide with easy to follow, step-by-step Getting Started and Tutorial.
- Full schematics

Samples:

The supplied device is typically a PIC17C756A.

SALES AND SUPPORT

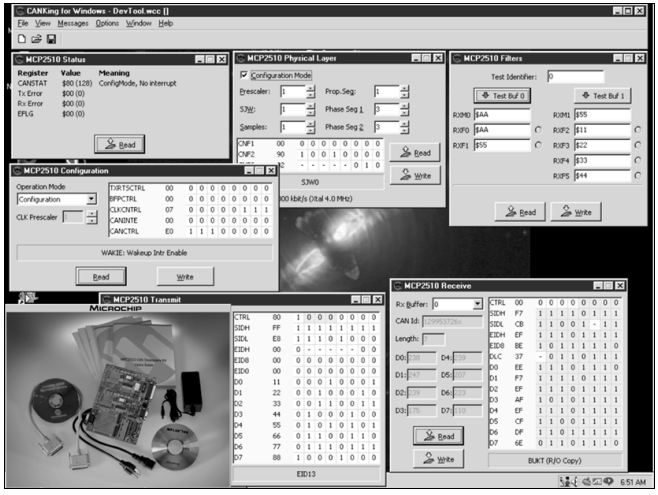
To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
DM173001	PICDEM-17

NOTES:

MCP2510

CAN Development Kit



The MCP2510 Controller Area Network (CAN) Developer's Kit is ideal for CAN system developers as well as for CAN beginners.

MCP2510 software development is made easy by offering a variety of features to manipulate the functionality of the MCP2510. The MCP2510 CAN Developer's Kit provides the ability to read, display, and modify all registers of the MCP2510 on a

bit-by-bit or a byte-by-byte basis. Included on the target board are PICmicro® sockets, a header to access the required MCP2510 pins, and a prototype area for the user to quickly build and test his own CAN node. Also included are on-board transceivers that have jumper-configureable options to allow different bus setups. In addition, this tool provides the user with an expansion connector for connecting a user-created CAN network. By using this expansion connector in this manner, the PC interface can be used as a simple bus monitor for CAN message traffic.

For CAN beginners, the MCP2510 CAN Developer's Kit can be used as a low-cost method of demonstrating basic input and output functionality by transmitting and receiving CAN messages. Transmitted messages are set up via an easy-to-use Windows® interface. LEDs connected to the MCP2510 transmit and receive pins toggle to show message traffic. Both analog and digital signals can be generated on the target board. These signals are then received by the host PC and displayed in a de-stuffed format for easy identification of message contents. In this manner, basic CAN communication can be demonstrated and understood.

Features:

- On-board features speed understanding:
  - Ability to read, display, and modify all registers
  - Ability to manipulate message mask and message filter functions
  - Modifications can be done on a bit-by-bit basis or a byte-by-byte basis
- Aids in development of users' CAN network:
  - On-board industry-standard CAN transceivers
  - Prototype area for user-defined transceivers that are jumper selectable
  - Expansion connector enables users to connect external CAN network and use PC interface as a basic bus monitor
  - PICmicro sockets, access to MCP2510 signals and prototype area for quick CAN mode development
- CAN messages demonstration capability for CAN beginners:
  - Easy to create and send CAN messages
  - Displays received messages in de-stuffed format
  - Target board contains switches and dials to vary message contents
  - LEDs toggle on and off to signify CAN message traffic
  - Familiar user interface

Ordering Information:

See the Microchip Development Systems Ordering Guide (DS30177) or www.microchip.com for specific part numbers. To order or obtain more information about the MCP2510 CAN Developer's Kit or any other Microchip product, contact the Microchip Sales Office, representative, or distributor nearest you.

Host System Requirements:

PC with 486 or higher processor. Pentium® recommended

4 MB Memory, 8 MB recommended

2 MB hard disk space, 5 MB recommended

VGA or Super VGA Monitor

Microsoft® Windows 3.1 or greater

Parallel Port

Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers, and Corporate Application Engineers. Microchip's Internet home page can be reached at: www.microchip.com

System Description:

The KEELOQ Transponder Evaluation Kit Hardware consists of a base station, a transmitter/transponder, a batteryless transponder, and various HCS410 samples. The base station doubles as a programmer and decoder. The base station includes a coil used for generating a magnetic field used to communicate with a transponder inductively. The base station has an RF receiver for receiving KeeLoq code hopping transmissions.

The accompanying Windows software is supplied on a 3.5-inch diskette and includes all the necessary software for programming and testing the HCS410 in all its modes.

To order or obtain more information about KeeLoq or any other Microchip product, contact the Microchip sales office nearest you.

Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers, and Corporate Application Engineers. Microchip's Internet home page can be reached at: www.microchip.com

Development Tools from Microchip	
MPLAB™	Integrated Development Environment (IDE)
MPLAB-C17	C compiler for PIC17CXXX MCUs
MPLAB-C18	C compiler for PIC18CXXX MCUs
C compiler	Sold by third-party vendors (HI-TECH, IAR, CCS)
MPASM	Universal PICmicro macro-assembler
MPLINK/MPLIB	Linker/Librarian
MPLAB-SIM	Software Simulator
MPLAB-ICD	In-circuit debugger evaluation kit
MPLAB-ICE 2000	Full-featured modular in-circuit emulator
ICEPIC	Low-cost in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART®Plus	Entry-level development kit with programmer

microID™

Programmer Kit

DESCRIPTION

The microID Programmer Kit is used to contactlessly program MCRF200 or MCRF250 microID devices. The programmer is calibrated for ISO-card tags but can be adjusted to be used for virtually any 125 kHz microID tag configuration, including button tags and keyfobs.

The microID Programmer's Kit includes:

1. RF-LAB 125 Software Interface (runs under Windows 95/98)
2. Contactless Programmer
3. Power Supply (110/220V)
4. RS-232 Serial Cable
5. Documentation including Microchip's Technical Library CD-ROM and a complete *125 kHz System Design Guide* (application notes, designs, and tutorials)

Order Information:

Description	Part Number
microID Programmer's Kit	PG103001

NOTES:

125 kHz microID Developer's Kit

DESCRIPTION

The 125 kHz microID Developer's Kit is an easy-to-use tool for design engineers at all skill levels. This kit includes all the hardware, software, reference designs, and samples required to get started in RFID designs.

The developer's kit also includes a contactless programmer and readers for three different configurations of MCRF200: 123h (ASK), 08Dh (PSK), and 00Ah (FSK).

The programmer is used to contactlessly program MCRF200 or MCRF250 microID devices. The programmer is calibrated for ISO-card tags but can be adjusted to be used for virtually any 125 kHz microID tag configuration, including button tags and keyfobs.

FEATURES

The 125 kHz microID Developer's Kit includes:

1. PSK Reader
2. FSK Reader
3. ASK Reader
4. Contactless Programmer
5. Power Supplies (2)
6. RS-232 Cables (2)
7. RF-LAB 125 Software Interface (runs under Windows 95/98)
8. Samples in Card-Tag Form (123h, 08Dh, 00Ah)
9. Samples in DIP Form (123h, 08Dh, 00Ah)
10. Documentation including Microchip's Technical Library CD-ROM and a complete 1*3.56 MHz System Design Guide* (application notes, reference designs, and tutorials)

Order Information:

Description	Part Number
125 kHz microID Developer's Kit	DV103001

125 kHz Anticollision microID Developer's Kit

DESCRIPTION

The 125 kHz Anticollision microID Developer's Kit is an easy-to-use tool for design engineers at all skill levels. This kit includes all the hardware, software, reference designs, and samples required to get started in 125 kHz anticollision (multiread) RFID designs.

The developer's kit also includes a contactless programmer and anticollision reader for FSK configuration of MCRF250 (40Ah).

The Programmer is used to contactlessly program MCRF200 or MCRF250 microID devices. The programmer is calibrated for ISO-card tags but can be adjusted to be used for virtually any 125 kHz microID tag configuration, including button tags and keyfobs.

FEATURES

The 125 kHz Anticollision microID Developer's Kit includes:

1. FSK Anticollision Reader
2. Contactless Programmer
3. Power Supplies (2)
4. RS-232 Cables (2)
5. RF-LAB 125 Software Interface (runs under Windows 95/98)
6. Samples in Card and DIP Form
7. Documentation including Microchip's Technical Library CD-ROM and a complete 1*25 kHz System Design Guide* (application notes, designs, and tutorials

Order Information:

Description	Part Number
125 kHz Anticollision microID Developer's Kit	DV103002

13.56 MHz Anticollision microID Developer's Kit

DESCRIPTION

The 13.56 MHz microID Developer's Kit is an easy-to-use tool for design engineers at all skill levels. This kit includes all the hardware, software, reference designs, and samples required to get started in 13.56 MHz RFID designs.

This kit is intended to show basic operation of the high-performance MCRF355 tagging chip.

The 13.56 MHz Anticollision microID Developer's Kit includes:

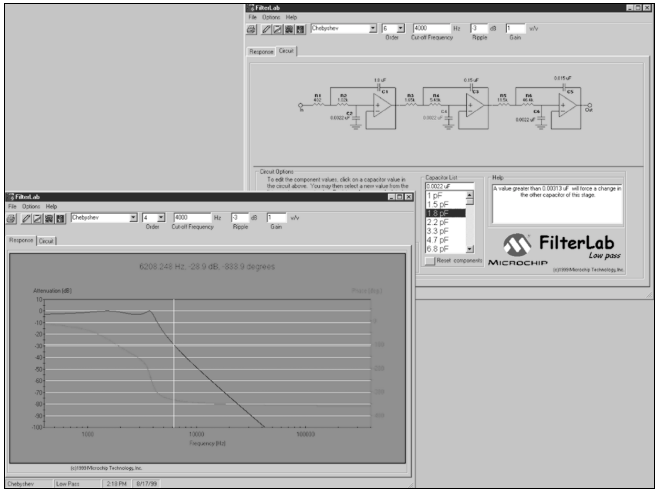
1. 13.56 MHz Anticollision Reader
2. Contact Programmer
3. Power Supplies (2)
4. RS-232 Cables (2)
5. RF-LAB 13.56 Software Interface (runs under Windows 95/98)
6. Socketed Tags
7. Flexible, Preprogrammed Performa™ tags by Checkpoint Systems Inc.
8. Samples in DIP Form
9. Documentation including Microchip's Technical Library CD-ROM and a complete *13.56 MHz System Design Guide*

Order Information:

Description	Part Number
13.56 MHz Anticollision microID Developer's Kit	DV103003

FILTERLAB™

Active Filter Software Design Tool



The difficult job of low-pass, active filter design is made easy with FilterLab software.

FilterLab is an innovative software tool that simplifies active filter design. Available at no cost from Microchip's web site (www.microchip.com), the FilterLab active filter software design tool provides full schematic diagrams of the filter circuit with component values and displays the frequency response.

FilterLab allows the design of low-pass filters up to an 8th order filter with Chebyshev, Bessel or Butterworth responses from frequencies of 0.1 Hz to 10 MHz. Users can select a flat passband or sharp transition from passband to stopband. Options, such as minimum ripple factor, sharp transition and linear phase delay, are available. Once the filter response has been identified, FilterLab generates the frequency response and the circuit. For maximum design flexibility, changes in capacitor values can be implemented to fit the demands of the application. FilterLab will recalculate all values to meet the desired response, allowing real-world values to be substituted or changed as part of the design process.

FilterLab also generates a spice model of the designed filter. Extraction of this model will allow time domain analysis in spice simulations, streamlining the design process.

Further consideration is given to designs used in conjunction with an analog-to-digital converter. A suggested filter can be generated by simply inputting the bit resolution and sample rate via the Anti-Aliasing Wizard. This eliminates erroneous signals folded back into the digital data due to the aliasing effect.

Features:

- Multiple Filter Order and Responses with Gain Option
  - Ability to select Bessel, Butterworth or Chebyshev filter response
  - Up to 8th-order filters can be simulated
  - Circuit diagram and component values given
- Bode Plot with Phase Margin
  - –Resultant Bode plot generated
- Circuit Implementation
  - Standard 1 percent resistors
  - Standard capacitor values generate and user adjustable
  - Circuit configuration: Sallen-Key (noninverting) or multiple feedback (inverting)
- Spice Model Generated
  - Spice Model of entire filter generated
  - Allows for streamline of simulations
- Anti-Aliasing Wizard
  - Filter optimization for Analog-to-Digital Converter base on bit resolution and sample rate

Ordering Information:

FilterLab

Devices Supported:

PC

Host System Requirements:

PC with 386 or higher processor.
Pentium® recommended

8 MB Memory, 32 MB recommended

16 MB hard disk space,
20 MB recommended

600 x 800 Monitor

Microsoft® Windows® 95/98

CD-ROM Drive

Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers, and Corporate Application Engineers. Microchip's Internet home page can be reached at: www.microchip.com

Development Tools from Microchip	
MPLAB™	Integrated Development Environment (IDE)
MPLAB-C17	C compiler for PIC17CXXX MCUs
MPLAB-C18	C compiler for PIC18CXXX MCUs
C compiler	Sold by third-party vendors (HI-TECH, IAR, CCS)
MPASM	Universal PICmicro macro-assembler
MPLINK/MPLIB	Linker/Librarian
MPLAB-SIM	Software Simulator
MPLAB-ICD	In-circuit debugger evaluation kit
MPLAB-ICE 2000	Full-featured modular in-circuit emulator
ICEPIC	Low-cost in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART®Plus	Entry-level development kit with programmer

SEEVAL® DESIGNER'S KIT

Microchip Serial EEPROM Designer's Kit

FEATURES

- Includes everything necessary to begin developing Serial EEPROM-based applications
- Microchip *Total Endurance*™ software model
- Microchip *SEEVAL* evaluation and programming board
- Microchip *SEEVAL* software
- Microchip Serial EEPROM handbook
- Microchip Serial EEPROM sample pack
- RS-232 serial cable
- Power supply

SYSTEM REQUIREMENTS

- DOS 3.1 or higher
- Windows® 3.1
- VGA monitor
- 386 or 486 processor recommended
- Math coprocessor recommended

DEVICE SUPPORT

- Microchip 2-wire 24CXX/24LCXXB/85CXX
- Microchip *Smart Serial*™ 24XX65
- Microchip 3-wire 93CXX/93LCXX series
- Microchip 4-wire 59C11

SEEVAL® Designer's Kit

DESCRIPTION

Now designers of Serial EEPROM-based applications can enjoy the increased productivity, reduced time to market, and the ability to create a rock-solid design that only a well-thought-out development system can provide. Microchip's Serial EEPROM Designer's Kit includes everything necessary to quickly develop a robust and reliable Serial EEPROM-based design and greatly reduce the time required for system integration and hardware/software debug.

The **Total Endurance software model** enables designers to quickly choose the best Serial EEPROM for the specific application and perform trade-off analysis with voltage, temperature, write cycle and other system parameters in order to achieve the desired Erase/Write endurance (specific ppm rate) or product lifetime. Total Endurance is the new standard of excellence in understanding and predicting the Erase/Write endurance of Serial EEPROMs. An on-line endurance tutorial is included, along with hypertext help files.

Microchip's **SEEVAL Serial EEPROM evaluation and programming system** will accept any Microchip Serial EEPROM in DIP package and enable the designer or system integrator to read, write, or erase any byte or the entire array. SEEVAL also provides the following advanced features to aid in system integration and debug:

- Program special user functions like *Smart Serial* configurations
- Hexadecimal display of array contents
- Pre-set or user-defined repeating patterns
- User-configurable functions like continuous read/write, programmable delay, etc.
- Upload/download files between the Serial EEPROM and disk

Another industry first, the **Microchip Serial EEPROM Handbook** provides a plethora of information crucial to the designers of Serial EEPROM-based systems. Along with data sheets on Microchip Serial EEPROMs, this resource provides application notes regarding Erase/Write endurance, interfacing with different protocols and many, many others. A cross-reference and selector guide are also included, plus article reprints and qualification reports on Microchip Serial EEPROMs.

USING SEEVAL AND TOTAL ENDURANCE

Both software packages can be loaded from Windows by choosing FILE RUN and entering SETUP.EXE from the Program Manager. The applications will install themselves; then a double mouse-click will start either application. The first step in either program is to select a device from the device list.

In Total Endurance, the user has simply to choose a Microchip Serial EEPROM device from the device-list menu and begin entering the application parameters. The entire process can take literally seconds to complete, and the model will output the PPM level and FIT rate of the device vs. the number of Erase/Write cycles. If the user has specified an application lifetime, the model will output PPM and FIT rates at that point in time. Alternately, the user may input a desired PPM level and the model will calculate the application lifetime which will result in that survival rate. The user may then trade-off any of the parameters (device type, voltage, application life, temperature, # of bytes per cycle, # of cycles per day etc.) to arrive at an optimal solution for the intended application.

Whenever a parameter is changed, calculation of the ppm/application life is automatic. An "update" box will appear inside the graph to indicate that new data has been entered and the graph should be redrawn. A single click in the "draw" box will redraw the plot of ppm vs. cycles; a click in the "Resize" box will take the plot to full-screen display for a closer view. The plot data can be saved to a file or the plot itself can be copied to the clipboard to be pasted into another application.

In **SEEVAL**, the user may choose to load a file from disk to program the Serial EEPROM, or read data from the EEPROM and save it to disk. The screen displays the contents of a software buffer. The buffer may be manipulated before programming data to the Serial EEPROM, or data can be written to the Serial EEPROM directly on-line. An area of memory can be highlighted (selected) and programmed with a predefined pattern or user-specified pattern. Alternately, the entire device can be programmed with any repeating pattern.

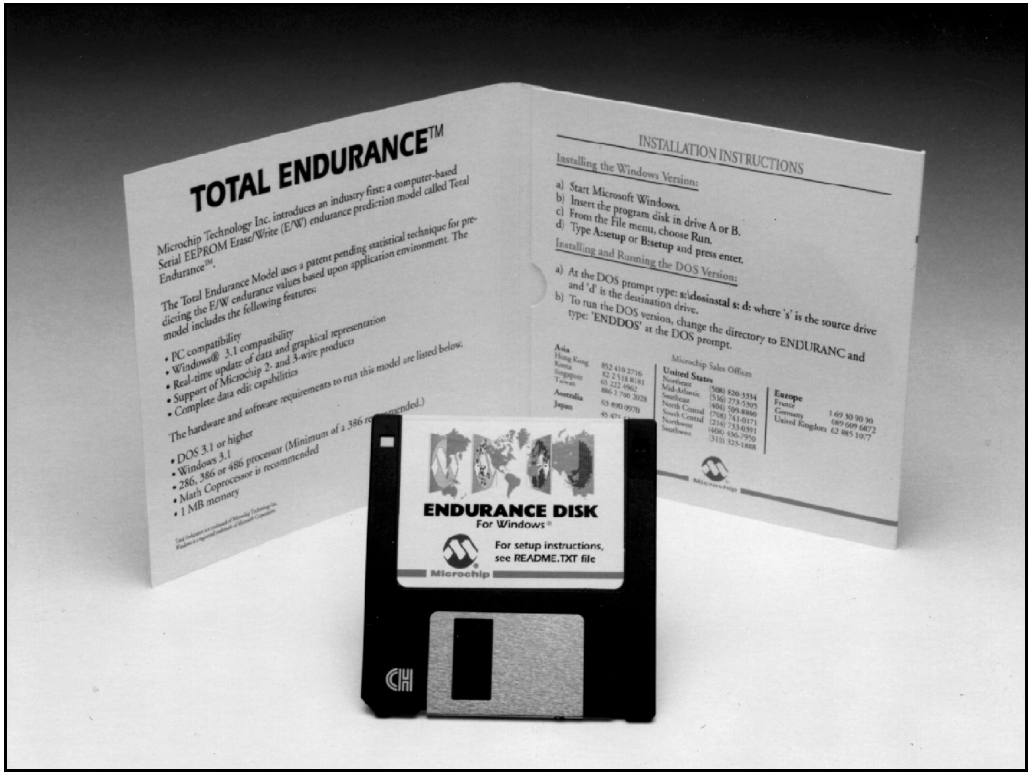
Both SEEVAL and Total Endurance allow the user to save any configuration as default. This configuration (device and application settings) will then automatically load at boot time.

Order Information:

Description	Part Number
Serial EEPROM Designer's Kit	DV243001

MICROCHIP TOTAL ENDURANCE™

Microchip Serial EEPROM Endurance Model



FEATURES

- IBM® PC compatibility
- Windows® 3.1 or DOS 3.1 compatibility
- Automatic or manual recalculation
- Real-time update of data
- Full-screen or windowed graphical view
- Hypertext on-screen help
- Key or slide-bar entry of parameters
- On-screen editing of parameters
- Single-click copy of plot to clipboard
- Numeric export to delimited text file
- On-disk Endurance Tutorial

SYSTEM REQUIREMENTS

- DOS 3.1 or higher
- Windows 3.1
- 1MB memory
- 386 or 486 processor recommended
- Math coprocessor recommended

DEVICE SUPPORT

- Microchip 2-wire 24CXX/24LCXXB/24AAXX/85CXX
- Microchip 3-wire 93CXX/93LCXX/93AAXX Series
- Microchip 4-wire 59C11

Total Endurance™

DESCRIPTION

Microchip's revolutionary Total Endurance Model provides electronic systems designers with unprecedented visibility into Serial EEPROM-based applications. This advanced software model (with a very friendly user interface) eliminates time and guesswork from Serial EEPROM-based designs by accurately predicting the device's performance and reliability within a user-defined application environment. Design trade-off analysis which formerly consumed days or weeks can now be performed in minutes...with a level of accuracy that delivers a truly robust design.

Users may input the following application parameters:

- Serial EEPROM device type
- Bytes to be written per cycle
- Cycling mode - byte or page
- Data pattern type - random or worst-case
- Temperature in °C
- Erase/Write cycles per day
- Application lifetime or target PPM level

The model will respond with FIT rate, PPM level, application life and a plot of the PPM level vs. number of cycles. The model is available in both DOS and Windows versions.

BACKGROUND

Microchip's research into the Erase/Write endurance of Serial EEPROMs has resulted in the conclusion that endurance depends upon three primary effects: the physical properties of the EEPROM cell, the internal error-correction technology employed, and the application environment. EEPROM endurance specified as a "typical" value in device data sheets must therefore be evaluated on a case-by-case basis, taking into account the manner in which the device will be used in the application. The Microchip Total Endurance™ software applies the user-defined application parameters to a complex mathematical model in order to emulate the EEPROM's performance and reliability in the system.

USING THE MODEL

The user has simply to choose a Microchip Serial EEPROM device from the device-list menu and begin entering the application parameters. The entire process can take literally seconds to complete, and the model will output the PPM level and FIT rate of the device vs. the number of Erase/Write cycles. If the user has specified an application lifetime, the model will output PPM and FIT rates at that point in time. Alternately, the user may input a desired PPM level and the model will calculate the application lifetime which will result in that survival rate. The user may then trade-off any of the parameters (device type, voltage, application life, temperature, # of bytes per cycle, # of cycles per day etc.) to arrive at an optimal solution for the intended application.

Whenever a parameter is changed, calculation of the ppm/application life is automatic. An "update" box will appear inside the graph to indicate that new data has been entered and the graph should be redrawn. A single click in the "draw" box will redraw the plot of ppm vs. cycles; a click in the "Resize" box will take the plot to full-screen display for a closer view. The plot data can be saved to a file or the plot itself can be copied to the clipboard to be pasted into another application.

ACCURACY OF THE MODEL

The accuracy of the Microchip Total Endurance model has been verified against test data to within ten percent of the actual values. However, Microchip makes no warranty as to its accuracy or applicability of the information to any given application. It is intended to be used as a guide to aid designers of Serial EEPROM-based systems in performing trade-off analysis and

Order Information:

Description	Part Number
Total Endurance Software Disk	SW242001

NOTES:

NOTES:

NOTES:

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-786-7200 Fax: 480-786-7277
Technical Support: 480-786-7627
Web Address: http://www.microchip.com

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
Microchip Technology Inc.
4570 Westgrove Drive, Suite 160
Addison, TX 75248
Tel: 972-818-7423 Fax: 972-818-2924

Dayton
Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Detroit
Microchip Technology Inc.
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

AMERICAS (continued)

Toronto
Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong
Microchip Asia Pacific
Unit 2101, Tower 2
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

Beijing
Microchip Technology, Beijing
Unit 915, 6 Chaoyangmen Bei Dajie
Dong Erhuan Road, Dongcheng District
New China Hong Kong Manhattan Building
Beijing 100027 PRC
Tel: 86-10-85282100 Fax: 86-10-85282104

India
Microchip Technology Inc.
India Liaison Office
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Japan
Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa 222-0033 Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Shanghai
Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hong Qiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700 Fax: 86 21-6275-5060

ASIA/PACIFIC (continued)

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan, R.O.C
Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5858 Fax: 44-118 921-5835

Denmark
Microchip Technology Denmark ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Arizona Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - Ier Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 München, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

11/15/99

DNV Certification, Inc. USA
ANSI-RAB QMS ACCREDITED
DNV
DNV MSC The Netherlands Accredited by the RvA
ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.