Microcontrollers

C166 Family

16-Bit Single-Chip Microcontroller

C164

User's Manual   1999-09

V 2.0

Edition 1999-09

Published by Infineon Technologies AG,
St.-Martin-Strasse 53
D-81541 München

Attention please!

The information herein is given to describe certain components and shall not be considered as warranted characteristics.
Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Infineon Technologies is an approved CECC manufacturer.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office in Germany or our Infineon Technologies Representatives worldwide (see address list).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Microcontrollers

C166 Family

16-Bit Single-Chip Microcontroller

C164

V 2.0, 1999-09

User's Manual

C164
Revision History: **1999-09 (V 2.0)**

Previous Versions: User’s Manual C164CI[1], 1998-08 (V1.1, still valid for OTP version)
User’s Manual C164CI[1], 11.97 (V 1.0)

Page	**Subjects** (major changes since last revision)
3-9, -10	XRAM description added
3-14...40	Flash memory description added
7-3...8	Port driver control introduced
7-25, -28	Open drain mode for Port 4 introduced
7-30	Register P5DIDIS introduced
7-26, -27, 7-33, -34, 19-10, 19-35...37	Programmable Interface Routing introduced
13-1ff	Additional prescaler introduced
17-1ff	Description of CAPCOM6 improved
19-7ff	Clock prescaler introduced
20-20ff	True Single-Chip mode reset introduced
21-5, -6	Sleep mode introduced
21-16...19	Programmable frequency output introduced

[1] OTP version only.

Controller Area Network (CAN): License of Robert Bosch GmbH

We Listen to Your Comments
Any information within this document that you feel is wrong, unclear or missing at all?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:
mcdocu.comments@infineon.com

This manual describes the Flash versions (C164xy-8F) and the ROM versions (C164xy-8R).

Table of Contents

Table of Contents Page

Table of Contents **Page**

Table of Contents Page

Table of Contents Page

Table of Contents Page

1 Introduction

The rapidly growing area of embedded control applications is representing one of the most time-critical operating environments for today's microcontrollers. Complex control algorithms have to be processed based on a large number of digital as well as analog input signals, and the appropriate output signals must be generated within a defined maximum response time. Embedded control applications also are often sensitive to board space, power consumption, and overall system cost.

Embedded control applications therefore require microcontrollers, which...

- offer a high level of system integration
- eliminate the need for additional peripheral devices and the associated software overhead
- provide system security and fail-safe mechanisms
- provide effective means to control (and reduce) the device's power consumption.

With the increasing complexity of embedded control applications, a significant increase in CPU performance and peripheral functionality over conventional 8-bit controllers is required from microcontrollers for high-end embedded control systems. In order to achieve this high performance goal Infineon has decided to develop its family of 16-bit CMOS microcontrollers without the constraints of backward compatibility.

Of course the architecture of the 16-bit microcontroller family pursues successfull hardware and software concepts, which have been established in Infineon's popular 8-bit controller families.

About this Manual

This manual describes the functionality of a number of 16-bit microcontrollers of the Infineon C166 Family, the C164-class.

As these microcontrollers provide a great extent of identical functionality it makes sense to describe a superset of the provided features. For this reason some sections of this manual do not refer to all the C164 derivatives that are offered or planned (e.g. devices with different kinds of on-chip memory or peripherals).

The descriptions in this manual refer to the following derivatives of the C164-class:

• **C164CI-8RM**, **C164CH-8FM**	Version with full function CAPCOM6 and CAN module
• **C164SI-8RM**, **C164SH-8FM**	Version with full function CAPCOM6
• **C164CL-8RM**	Version with reduced CAPCOM6 and CAN module
• **C164SL-8RM**	Version with reduced CAPCOM6

This manual is valid for the mentioned derivatives. Of course it refers to all devices of the different available temperature ranges and packages.

For simplicity all these various versions are referred to by the term **C164** throughout this manual. The complete pro-electron conforming designations are listed in the respective data sheets.

1.1 The Members of the 16-bit Microcontroller Family

The microcontrollers of the Infineon 16-bit family have been designed to meet the high performance requirements of real-time embedded control applications. The architecture of this family has been optimized for high instruction throughput and minimum response time to external stimuli (interrupts). Intelligent peripheral subsystems have been integrated to reduce the need for CPU intervention to a minimum extent. This also minimizes the need for communication via the external bus interface. The high flexibility of this architecture allows to serve the diverse and varying needs of different application areas such as automotive, industrial control, or data communications.

The core of the 16-bit family has been developped with a modular family concept in mind. All family members execute an efficient control-optimized instruction set (additional instructions for members of the second generation). This allows an easy and quick implementation of new family members with different internal memory sizes and technologies, different sets of on-chip peripherals and/or different numbers of IO pins.

The XBUS concept opens a straight forward path for the integration of application specific peripheral modules in addition to the standard on-chip peripherals in order to build application specific derivatives.

As programs for embedded control applications become larger, high level languages are favoured by programmers, because high level language programs are easier to write, to debug and to maintain.

The 80C166-type microcontrollers were the **first generation** of the 16-bit controller family. These devices have established the C166 architecture.

The C165-type and C167-type devices are members of the **second generation** of this family. This second generation is even more powerful due to additional instructions for HLL support, an increased address space, increased internal RAM and highly efficient management of various resources on the external bus.

Enhanced derivatives of this second generation provide additional features like additional internal high-speed RAM, an integrated CAN-Module, an on-chip PLL, etc.

Utilizing integration to design efficient systems may require the integration of application specific peripherals to boost system performance, while minimizing the part count. These efforts are supported by the so-called XBUS, defined for the Infineon 16-bit microcontrollers (second generation). This XBUS is an internal representation of the external bus interface that opens and simplifies the integration of peripherals by standardizing the required interface. One representative taking advantage of this technology is the integrated CAN module.

The C165-type devices are reduced versions of the C167 which provide a smaller package and reduced power consumption at the expense of the A/D converter, the CAPCOM units and the PWM module.

The C164-type devices and some of the C161-type devices are further enhanced by a flexible power management and form the **third generation** of the 16-bit controller family. This power management mechanism provides effective means to control the power that is consumed in a certain state of the controller and thus allows the minimization of the overall power consumption with respect to a given application.

A variety of different versions is provided which offer various kinds of on-chip program memory:

- mask-programmable ROM
- Flash memory
- OTP memory
- ROMless with no non-volatile memory at all.

Also there are devices with specific functional units.

The devices may be offered in different packages, temperature ranges and speed classes.

More standard and application-specific derivatives are planned and in development.

Note: Not all derivatives will be offered in any temperature range, speed class, package or program memory variation.

Information about specific versions and derivatives will be made available with the devices themselves. Contact your Infineon representative for up-to-date material.

Note: As the architecture and the basic features (i.e. CPU core and built in peripherals) are identical for most of the currently offered versions of the C164, the descriptions within this manual that refer to the “C164” also apply to the other variations, unless otherwise noted.

1.2 Summary of Basic Features

The C164 is an improved representative of the Infineon family of full featured 16-bit single-chip CMOS microcontrollers. It combines high CPU performance (up to 12.5 million instructions per second) with high peripheral functionality and means for power reduction.
Several key features contribute to the high performance of the C164 (the indicated timings refer to a CPU clock of 25 MHz).

High Performance 16-Bit CPU With Four-Stage Pipeline

- 80 ns minimum instruction cycle time, with most instructions executed in 1 cycle
- 400 ns multiplication (16-bit *16-bit), 800 ns division (32-bit/16-bit)
- Multiple high bandwidth internal data buses
- Register based design with multiple variable register banks
- Single cycle context switching support
- 16 MBytes linear address space for code and data (von Neumann architecture)
- System stack cache support with automatic stack overflow/underflow detection

Control Oriented Instruction Set with High Efficiency

- Bit, byte, and word data types
- Flexible and efficient addressing modes for high code density
- Enhanced boolean bit manipulation with direct addressability of 6 Kbits for peripheral control and user defined flags
- Hardware traps to identify exception conditions during runtime
- HLL support for semaphore operations and efficient data access

Power Management Features

- Programmable system slowdown (slowdown divider SDD)
- Flexible peripheral management (individual disabling)
- Sleepmode including wakeup via external interrupts
- Programmable frequency output

Integrated On-chip Memory

- 64 KByte on-chip Program **Flash**[1] memory or Mask **ROM**
- 2 KByte internal RAM (IRAM) for variables, register banks, system stack and code
- 2 KByte on-chip high-speed extension RAM (XRAM) for variables, user stacks and code
- 4 KByte on-chip DataFlash/EEPROM[1] for non-volatile variables

[1] Available only on devices in Flash technology.

External Bus Interface

- Multiplexed or demultiplexed bus configurations
- Segmentation capability and chip select signal generation
- 8-bit or 16-bit data bus
- Bus cycle characteristics selectable for five programmable address areas

16-Priority-Level Interrupt System

- Up to 33 interrupt nodes with separate interrupt vectors
- 240/400 ns typical/maximum interrupt latency in case of internal program execution
- Fast external interrupts

8-Channel Peripheral Event Controller (PEC)

- Interrupt driven single cycle data transfer
- Transfer count option (std. CPU interrupt after programmable number of PEC transfers)
- Eliminates overhead of saving and restoring system state for interrupt requests

Intelligent On-chip Peripheral Subsystems

- 8-Channel 10-bit A/D Converter with programmable conversion time (7.76 µs minimum), auto scan modes, channel injection mode
- Two Multifunctional General Purpose Timer Units
  GPT1: three 16-bit timers/counters, maximum resolution $f_{CPU}/8$
  GPT2: two 16-bit timers/counters, maximum resolution $f_{CPU}/4$
- Two Capture/Compare Units with independent time bases each,
  very flexible PWM unit/event recording unit with different operating modes
- Asynchronous/Synchronous Serial Channel (USART)
  with baud rate generator, parity, framing, and overrun error detection
- High Speed Synchronous Serial Channel
  with programmable data length and shift direction
- CAN Module (2.0B active) with 15 Message Objects, Full-CAN/Basic-CAN
- Real Time Clock
- Watchdog Timer with programmable time intervals
- Bootstrap Loader for flexible system initialization

59 IO Lines With Individual Bit Addressability

- Tri-stated in input mode
- Selectable input thresholds (not on all pins)
- Push/pull or open drain output mode
- Programmable port driver control (fast/reduced edge)

Different Temperature Ranges

- 0 to +70 °C, – 40 to +85 °C, – 40 to +125 °C

Infineon CMOS Process

- Low Power CMOS Technology including power saving Idle, Sleep and Power Down modes with flexible power management.

80-Pin Plastic Metric Quad Flat Pack (MQFP) Package

- 0.65 mm (25.6 mil) lead spacing, surface mount technology

Complete Development Support

For the development tool support of its microcontrollers, Infineon follows a clear third party concept. Currently around 120 tool suppliers world-wide, ranging from local niche manufacturers to multinational companies with broad product portfolios, offer powerful development tools for the Infineon C500 and C166 microcontroller families, guaranteeing a remarkable variety of price-performance classes as well as early availability of high quality key tools such as compilers, assemblers, simulators, debuggers or in-circuit emulators.

Infineon incorporates its strategic tool partners very early into the product development process, making sure embedded system developers get reliable, well-tuned tool solutions, which help them unleash the power of Infineon microcontrollers in the most effective way and with the shortest possible learning curve.

The tool environment for the Infineon 16-bit microcontrollers includes the following tools:

- Compilers (C, MODULA2, FORTH)
- Macro-Assemblers, Linkers, Locaters, Library Managers, Format-Converters
- Architectural Simulators
- HLL debuggers
- Real-Time operating systems
- VHDL chip models
- In-Circuit Emulators (based on bondout or standard chips)
- Plug-In emulators
- Emulation and Clip-Over adapters, production sockets
- Logic Analyzer disassemblers
- Starter Kits
- Evaluation Boards with monitor programs
- Industrial boards (also for CAN, FUZZY, PROFIBUS, FORTH applications)
- Network driver software (CAN, PROFIBUS)

1.3 Abbreviations

The following acronyms and termini are used within this document:

ADC	Analog Digital Converter
ALE	Address Latch Enable
ALU	Arithmetic and Logic Unit
ASC	Asynchronous/synchronous Serial Controller
CAN	Controller Area Network (License Bosch)
CAPCOM	CAPture and COMpare unit
CISC	Complex Instruction Set Computing
CMOS	Complementary Metal Oxide Silicon
CPU	Central Processing Unit
EBC	External Bus Controller
EEPROM	Electrically Erasable/Programmable Read-Only Memory
ESFR	Extended Special Function Register
Flash	Non-volatile memory that may be electrically erased
GPR	General Purpose Register
GPT	General Purpose Timer unit
HLL	High Level Language
IO	Input / Output
OTP	One Time Programmable memory
PEC	Peripheral Event Controller
PLA	Programmable Logic Array
PLL	Phase Locked Loop
PWM	Pulse Width Modulation
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
ROM	Read Only Memory
SDD	Slow Down Divider
SFR	Special Function Register
SSC	Synchronous Serial Controller
XBUS	Internal representation of the External Bus

2 Architectural Overview

The architecture of the C164 combines the advantages of both RISC and CISC processors in a very well-balanced way. The sum of the features which are combined results in a high performance microcontroller, which is the right choice not only for today's applications, but also for future engineering challenges. The C164 not only integrates a powerful CPU core and a set of peripheral units into one chip, but also connects the units in a very efficient way. One of the four buses used concurrently on the C164 is the XBUS, an internal representation of the external bus interface. This bus provides a standardized method of integrating application-specific peripherals to produce derivatives of the standard C164.

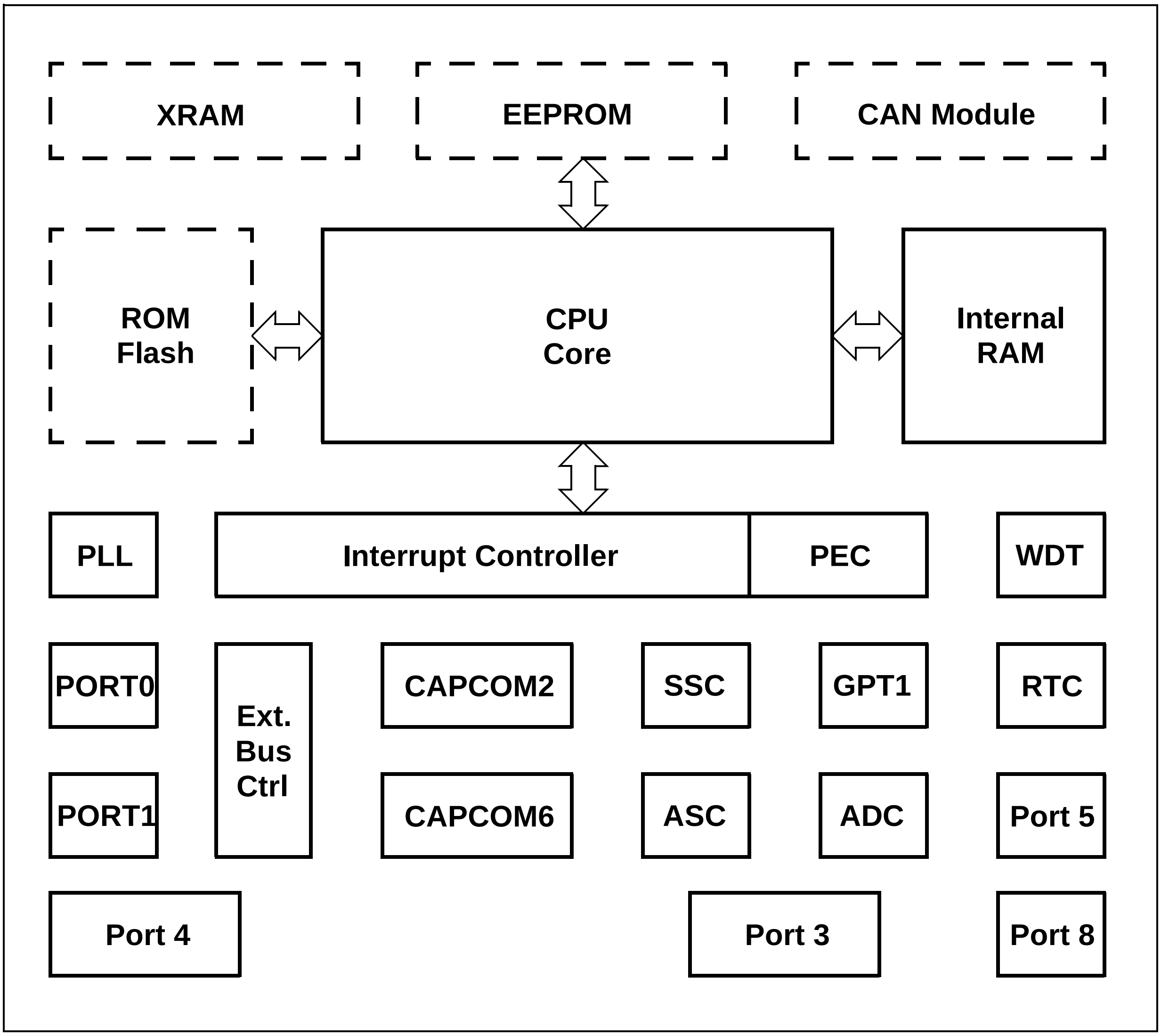


Figure 2-1 C164 Functional Block Diagram

2.1 Basic CPU Concepts and Optimizations

The main core of the CPU consists of a 4-stage instruction pipeline, a 16-bit arithmetic and logic unit (ALU) and dedicated SFRs. Additional hardware is provided for a separate multiply and divide unit, a bit-mask generator and a barrel shifter.

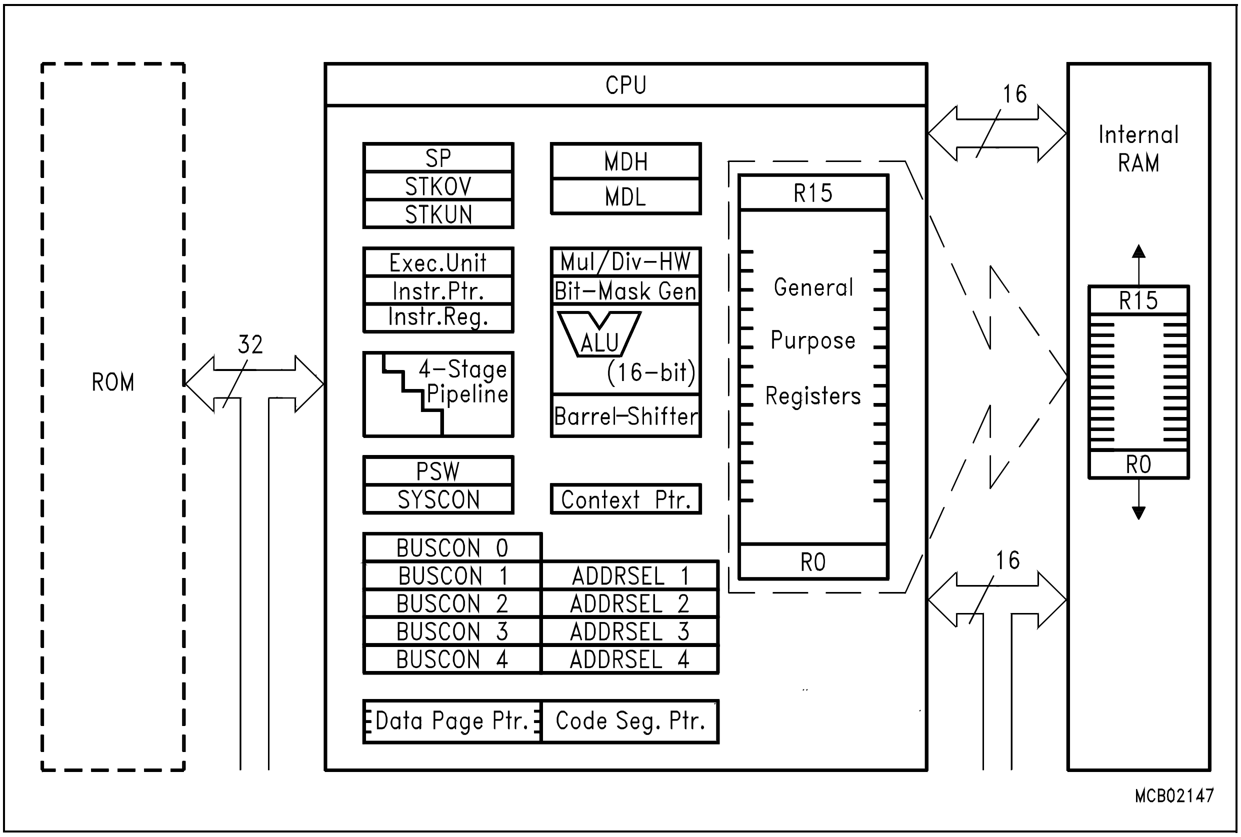


Figure 2-2 CPU Block Diagram

To meet the demand for greater performance and flexibility, a number of areas has been optimized in the processor core. Functional blocks in the CPU core are controlled by signals from the instruction decode logic. These are summarized below, and described in detail in the following sections:

1) High Instruction Bandwidth / Fast Execution

2) High Function 8-bit and 16-bit Arithmetic and Logic Unit

3) Extended Bit Processing and Peripheral Control

4) High Performance Branch-, Call-, and Loop Processing

5) Consistent and Optimized Instruction Formats

6) Programmable Multiple Priority Interrupt Structure

2.1.1 High Instruction Bandwidth / Fast Execution

Based on the hardware provisions, most of the C164's instructions can be executed in just one machine cycle, which requires 2 CPU clock cycles (2 * 1/f_{CPU} = 4 TCL). For example, shift and rotate instructions are always processed within one machine cycle, independent of the number of bits to be shifted.

Branch-, multiply- and divide instructions normally take more than one machine cycle. These instructions, however, have also been optimized. For example, branch instructions only require an additional machine cycle, when a branch is taken, and most branches taken in loops require no additional machine cycles at all, due to the so-called 'Jump Cache'.
A 32-bit / 16-bit division takes 20 CPU clock cycles, a 16-bit * 16-bit multiplication takes 10 CPU clock cycles.

The instruction cycle time has been dramatically reduced through the use of instruction pipelining. This technique allows the core CPU to process portions of multiple sequential instruction stages in parallel. The following four stage pipeline provides the optimum balancing for the CPU core:

FETCH: In this stage, an instruction is fetched from the internal ROM or RAM or from the external memory, based on the current IP value.

DECODE: In this stage, the previously fetched instruction is decoded and the required operands are fetched.

EXECUTE: In this stage, the specified operation is performed on the previously fetched operands.

WRITE BACK: In this stage, the result is written to the specified location.

If this technique were not used, each instruction would require four machine cycles. This increased performance allows a greater number of tasks and interrupts to be processed.

Instruction Decoder

Instruction decoding is primarily generated from PLA outputs based on the selected opcode. No microcode is used and each pipeline stage receives control signals staged in control registers from the decode stage PLAs. Pipeline holds are primarily caused by wait states for external memory accesses and cause the holding of signals in the control registers. Multiple-cycle instructions are performed through instruction injection and simple internal state machines which modify required control signals.

High Function 8-bit and 16-bit Arithmetic and Logic Unit

All standard arithmetic and logical operations are performed in a 16-bit ALU. In addition, for byte operations, signals are provided from bits six and seven of the ALU result to correctly set the condition flags. Multiple precision arithmetic is provided through a ’CARRY-IN’ signal to the ALU from previously calculated portions of the desired operation.

Most internal execution blocks have been optimized to perform operations on either 8-bit or 16-bit quantities. Once the pipeline has been filled, one instruction is completed per machine cycle, except for multiply and divide. An advanced Booth algorithm has been incorporated to allow four bits to be multiplied and two bits to be divided per machine cycle. Thus, these operations use two coupled 16-bit registers, MDL and MDH, and require four and nine machine cycles, respectively, to perform a 16-bit by 16-bit (or 32-bit by 16-bit) calculation plus one machine cycle to setup and adjust the operands and the result. Even these longer multiply and divide instructions can be interrupted during their execution to allow for very fast interrupt response. Instructions have also been provided to allow byte packing in memory while providing sign extension of bytes for word wide arithmetic operations. The internal bus structure also allows transfers of bytes or words to or from peripherals based on the peripheral requirements.

A set of consistent flags is automatically updated in the PSW after each arithmetic, logical, shift, or movement operation. These flags allow branching on specific conditions. Support for both signed and unsigned arithmetic is provided through user-specifiable branch tests. These flags are also preserved automatically by the CPU upon entry into an interrupt or trap routine.
All targets for branch calculations are also computed in the central ALU.

A 16-bit barrel shifter provides multiple bit shifts in a single cycle. Rotates and arithmetic shifts are also supported.

Extended Bit Processing and Peripheral Control

A large number of instructions has been dedicated to bit processing. These instructions provide efficient control and testing of peripherals while enhancing data manipulation. Unlike other microcontrollers, these instructions provide direct access to two operands in the bit-addressable space without requiring to move them into temporary flags.

The same logical instructions available for words and bytes are also supported for bits. This allows the user to compare and modify a control bit for a peripheral in one instruction. Multiple bit shift instructions have been included to avoid long instruction streams of single bit shift operations. These are also performed in a single machine cycle.

In addition, bit field instructions have been provided, which allow the modification of multiple bits from one operand in a single instruction.

High Performance Branch-, Call-, and Loop Processing

Due to the high percentage of branching in controller applications, branch instructions have been optimized to require one extra machine cycle only when a branch is taken. This is implemented by precalculating the target address while decoding the instruction. To decrease loop execution overhead, three enhancements have been provided:

- The first solution provides single cycle branch execution after the first iteration of a loop. Thus, only one machine cycle is lost during the execution of the entire loop. In loops which fall through upon completion, no machine cycles are lost when exiting the loop. No special instructions are required to perform loops, and loops are automatically detected during execution of branch instructions.

- The second loop enhancement allows the detection of the end of a table and avoids the use of two compare instructions embedded in loops. One simply places the lowest negative number at the end of the specific table, and specifies branching if neither this value nor the compared value have been found. Otherwise the loop is terminated if either condition has been met. The terminating condition can then be tested.

- The third loop enhancement provides a more flexible solution than the Decrement and Skip on Zero instruction which is found in other microcontrollers. Through the use of Compare and Increment or Decrement instructions, the user can make comparisons to any value. This allows loop counters to cover any range. This is particularly advantageous in table searching.

Saving of system state is automatically performed on the internal system stack avoiding the use of instructions to preserve state upon entry and exit of interrupt or trap routines. Call instructions push the value of the IP on the system stack, and require the same execution time as branch instructions.

Instructions have also been provided to support indirect branch and call instructions. This supports implementation of multiple CASE statement branching in assembler macros and high level languages.

Consistent and Optimized Instruction Formats

To obtain optimum performance in a pipelined design, an instruction set has been designed which incorporates concepts from Reduced Instruction Set Computing (RISC). These concepts primarily allow fast decoding of the instructions and operands while reducing pipeline holds. These concepts, however, do not preclude the use of complex instructions, which are required by microcontroller users. The following goals were used to design the instruction set:

1. Provide powerful instructions to perform operations which currently require sequences of instructions and are frequently used. Avoid transfer into and out of temporary registers such as accumulators and carry bits. Perform tasks in parallel such as saving state upon entry into interrupt routines or subroutines.
2. Avoid complex encoding schemes by placing operands in consistent fields for each instruction. Also avoid complex addressing modes which are not frequently used. This decreases the instruction decode time while also simplifying the development of compilers and assemblers.
3. Provide most frequently used instructions with one-word instruction formats. All other instructions are placed into two-word formats. This allows all instructions to be placed on word boundaries, which alleviates the need for complex alignment hardware. It also has the benefit of increasing the range for relative branching instructions.

The high performance offered by the hardware implementation of the CPU can efficiently be utilized by a programmer via the highly functional C164 instruction set which includes the following instruction classes:

- Arithmetic Instructions
- Logical Instructions
- Boolean Bit Manipulation Instructions
- Compare and Loop Control Instructions
- Shift and Rotate Instructions
- Prioritize Instruction
- Data Movement Instructions
- System Stack Instructions
- Jump and Call Instructions
- Return Instructions
- System Control Instructions
- Miscellaneous Instructions

Possible operand types are bits, bytes and words. Specific instruction support the conversion (extension) of bytes to words. A variety of direct, indirect or immediate addressing modes are provided to specify the required operands.

2.1.2 Programmable Multiple Priority Interrupt System

The following enhancements have been included to allow processing of a large number of interrupt sources:

1. Peripheral Event Controller (PEC): This processor is used to off-load many interrupt requests from the CPU. It avoids the overhead of entering and exiting interrupt or trap routines by performing single-cycle interrupt-driven byte or word data transfers between any two locations in segment 0 with an optional increment of either the PEC source or the destination pointer. Just one cycle is ’stolen’ from the current CPU activity to perform a PEC service.
2. Multiple Priority Interrupt Controller: This controller allows all interrupts to be placed at any specified priority. Interrupts may also be grouped, which provides the user with the ability to prevent similar priority tasks from interrupting each other. For each of the possible interrupt sources there is a separate control register, which contains an interrupt request flag, an interrupt enable flag and an interrupt priority bitfield. Once having been accepted by the CPU, an interrupt service can only be interrupted by a higher prioritized service request. For standard interrupt processing, each of the possible interrupt sources has a dedicated vector location.
3. Multiple Register Banks: This feature allows the user to specify up to sixteen general purpose registers located anywhere in the internal RAM. A single one-machine-cycle instruction allows to switch register banks from one task to another.
4. Interruptable Multiple Cycle Instructions: Reduced interrupt latency is provided by allowing multiple-cycle instructions (multiply, divide) to be interruptable.

With an interrupt response time within a range from just 5 to 10 CPU clock cycles (in case of internal program execution), the C164 is capable of reacting very fast on non-deterministic events.

Its fast external interrupt inputs are sampled every CPU clock cycle and allow to recognize even very short external signals.

The C164 also provides an excellent mechanism to identify and to process exceptions or error conditions that arise during run-time, so called ’Hardware Traps’. Hardware traps cause an immediate non-maskable system reaction which is similar to a standard interrupt service (branching to a dedicated vector table location). The occurrence of a hardware trap is additionally signified by an individual bit in the trap flag register (TFR). Except for another higher prioritized trap service being in progress, a hardware trap will interrupt any current program execution. In turn, hardware trap services can normally not be interrupted by standard or PEC interrupts.

Software interrupts are supported by means of the ’TRAP’ instruction in combination with an individual trap (interrupt) number.

2.2 The On-chip System Resources

The C164 controllers provide a number of powerful system resources designed around the CPU. The combination of CPU and these resources results in the high performance of the members of this controller family.

Peripheral Event Controller (PEC) and Interrupt Control

The Peripheral Event Controller allows to respond to an interrupt request with a single data transfer (word or byte) which only consumes one instruction cycle and does not require to save and restore the machine status. Each interrupt source is prioritized every machine cycle in the interrupt control block. If PEC service is selected, a PEC transfer is started. If CPU interrupt service is requested, the current CPU priority level stored in the PSW register is tested to determine whether a higher priority interrupt is currently being serviced. When an interrupt is acknowledged, the current state of the machine is saved on the internal system stack and the CPU branches to the system specific vector for the peripheral.

The PEC contains a set of SFRs which store the count value and control bits for eight data transfer channels. In addition, the PEC uses a dedicated area of RAM which contains the source and destination addresses. The PEC is controlled similar to any other peripheral through SFRs containing the desired configuration of each channel.

An individual PEC transfer counter is implicitly decremented for each PEC service except forming in the continuous transfer mode. When this counter reaches zero, a standard interrupt is performed to the vector location related to the corresponding source. PEC services are very well suited, for example, to move register contents to/from a memory table. The C164 has 8 PEC channels each of which offers such fast interrupt-driven data transfer capabilities.

Memory Areas

The memory space of the C164 is configured in a Von Neumann architecture which means that code memory, data memory, registers and IO ports are organized within the same linear address space which covers up to 16 MBytes. The entire memory space can be accessed bytewise or wordwise. Particular portions of the on-chip memory have additionally been made directly bit addressable.

A 2 KByte 16-bit wide internal RAM provides fast access to General Purpose Registers (GPRs), user data (variables) and system stack. The internal RAM may also be used for code. A unique decoding scheme provides flexible user register banks in the internal memory while optimizing the remaining RAM for user data.

The CPU has an actual register context consisting of up to 16 wordwide and/or bytewide GPRs at its disposal, which are physically located within the on-chip RAM area. A Context Pointer (CP) register determines the base address of the active register bank to be accessed by the CPU at a time. The number of register banks is only restricted by the available internal RAM space. For easy parameter passing, a register bank may overlap others.

A system stack of up to 1024 words is provided as a storage for temporary data. The system stack is also located within the on-chip RAM area, and it is accessed by the CPU via the stack pointer (SP) register. Two separate SFRs, STKOV and STKUN, are implicitly compared against the stack pointer value upon each stack access for the detection of a stack overflow or underflow.

Hardware detection of the selected memory space is placed at the internal memory decoders and allows the user to specify any address directly or indirectly and obtain the desired data without using temporary registers or special instructions.

A 2 KByte 16-bit wide on-chip XRAM provides fast access to user data (variables), user stacks and code. The on-chip XRAM is realized as an X-Peripheral and appears to the software as an external RAM. Therefore it cannot store register banks and is not bitaddressable. The XRAM allows 16-bit accesses with maximum speed.

For Special Function Registers 1024 Bytes of the address space are reserved. The standard Special Function Register area (SFR) uses 512 bytes, while the Extended Special Function Register area (ESFR) uses the other 512 bytes. (E)SFRs are wordwide registers which are used for controlling and monitoring functions of the different on-chip units. Unused (E)SFR addresses are reserved for future members of the C166 family with enhanced functionality.

An optional on-chip Flash or ROM memory provides for both code and constant data storage. This memory area is connected to the CPU via a 32-bit-wide bus. Thus, an entire double-word instruction can be fetched in just one machine cycle. The ROM will be mask programmed in the factory while the Flash memory can also be programmed within the application.

Program execution from on-chip program memory is the fastest of all possible alternatives.

The type of the on-chip program memory (Flash/ROM/none) depends on the chosen derivative.

A 4 KByte 16-bit wide on-chip DataFlash/EEPROM stores non-volatile user data. The on-chip EEPROM is realized as an X-Peripheral and appears to the software as external memory. Therefore it is not bitaddressable.

The EEPROM is organized in 4 sectors of 1 KByte each. Erasing can be done in units of bytes, words, pages (16 bytes), and sectors. Programming can be done in units of bytes, words, and pages (16 bytes).

External Bus Interface

In order to meet the needs of designs where more memory is required than is provided on chip, up to 4 MBytes of external RAM and/or ROM can be connected to the microcontroller via its external bus interface. The integrated External Bus Controller (EBC) allows to access external memory and/or peripheral resources in a very flexible way. For up to five address areas the bus mode (multiplexed / demultiplexed), the data bus width (8-bit/16-bit) and even the length of a bus cycle (waitstates, signal delays) can be selected independently. This allows to access a variety of memory and peripheral components directly and with maximum efficiency. If the device does not run in Single Chip Mode, where no external memory is required, the EBC can control external accesses in one of the following external access modes:

- 16-/18-/20-/22-bit Addresses, 16-bit Data, Demultiplexed
- 16-/18-/20-/22-bit Addresses, 8-bit Data, Demultiplexed
- 16-/18-/20-/22-bit Addresses, 16-bit Data, Multiplexed
- 16-/18-/20-/22-bit Addresses, 8-bit Data, Multiplexed

The demultiplexed bus modes use PORT1 for addresses and PORT0 for data input/output. The multiplexed bus modes use PORT0 for both addresses and data input/output. Port 4 is used for the upper address lines (A16..., if selected) and for the $\overline{\text{CS}}$ lines ($\overline{\text{CS}}$0..., if selected).

Important timing characteristics of the external bus interface (waitstates, ALE length and Read/Write Delay) have been made programmable to allow the user the adaption of a wide range of different types of memories and/or peripherals.

For applications which require less than 64 KBytes of address space, a non-segmented memory model can be selected, where all locations can be addressed by 16 bits, and thus Port 4 is not needed as an output for the upper address bits (Axx...A16), as is the case when using the segmented memory model.

The on-chip XBUS is an internal representation of the external bus and allows to access integrated application-specific peripherals/modules in the same way as external components. It provides a defined interface for these customized peripherals.

The on-chip XRAM and the on-chip CAN-Module are examples for these X-Peripherals.

2.3 The On-chip Peripheral Blocks

The C166 Family clearly separates peripherals from the core. This structure permits the maximum number of operations to be performed in parallel and allows peripherals to be added or deleted from family members without modifications to the core. Each functional block processes data independently and communicates information over common buses. Peripherals are controlled by data written to the respective Special Function Registers (SFRs). These SFRs are located either within the standard SFR area ($00'FE00_H$...$00'FFFF_H$) or within the extended ESFR area ($00'F000_H$...$00'F1FF_H$).

These built in peripherals either allow the CPU to interface with the external world, or provide functions on-chip that otherwise were to be added externally in the respective system.

The C164 generic peripherals are:

- A General Purpose Timer Block (GPT1)
- Two Serial Interfaces (ASC0 and SSC)
- A Watchdog Timer
- Two Capture / Compare units (CAPCOM2 and CAPCOM6)
- A 10-bit Analog / Digital Converter
- A Real Time Clock
- Six IO ports with a total of 59 IO lines

Each peripheral also contains a set of Special Function Registers (SFRs), which control the functionality of the peripheral and temporarily store intermediate data results. Each peripheral has an associated set of status flags. Individually selected clock signals are generated for each peripheral from binary multiples of the CPU clock.

Peripheral Interfaces

The on-chip peripherals generally have two different types of interfaces, an interface to the CPU and an interface to external hardware. Communication between CPU and peripherals is performed through Special Function Registers (SFRs) and interrupts. The SFRs serve as control/status and data registers for the peripherals. Interrupt requests are generated by the peripherals based on specific events which occur during their operation (e.g. operation complete, error, etc.).

For interfacing with external hardware, specific pins of the parallel ports are used, when an input or output function has been selected for a peripheral. During this time, the port pins are controlled by the peripheral (when used as outputs) or by the external hardware which controls the peripheral (when used as inputs). This is called the 'alternate (input or output) function' of a port pin, in contrast to its function as a general purpose IO pin.

Peripheral Timing

Internal operation of CPU and peripherals is based on the CPU clock (f_{CPU}). The on-chip oscillator derives the CPU clock from the crystal or from the external clock signal. The clock signal which is gated to the peripherals is independent from the clock signal which feeds the CPU. During Idle mode the CPU's clock is stopped while the peripherals continue their operation. Peripheral SFRs may be accessed by the CPU once per state. When an SFR is written to by software in the same state where it is also to be modified by the peripheral, the software write operation has priority. Further details on peripheral timing are included in the specific sections about each peripheral.

Programming Hints

Access to SFRs

All SFRs reside in data page 3 of the memory space. The following addressing mechanisms allow to access the SFRs:

- indirect or direct addressing with **16-bit (mem) addresses** must guarantee that the used data page pointer (DPP0...DPP3) selects data page 3.
- accesses via the Peripheral Event Controller (**PEC**) use the SRCPx and DSTPx pointers instead of the data page pointers.
- **short 8-bit (reg) addresses** to the standard SFR area do not use the data page pointers but directly access the registers within this 512 Byte area.
- **short 8-bit (reg) addresses** to the extended **ESFR** area require switching to the 512 Byte extended SFR area. This is done via the EXTension instructions EXTR, EXTP(R), EXTS(R).

Byte write operations to word wide SFRs via indirect or direct 16-bit (mem) addressing or byte transfers via the PEC force zeros in the non-addressed byte. Byte write operations via short 8-bit (reg) addressing can only access the low byte of an SFR and force zeros in the high byte. It is therefore recommended, to use the bit field instructions (BFLDL and BFLDH) to write to any number of bits in either byte of an SFR without disturbing the non-addressed byte and the unselected bits.

Reserved Bits

Some of the bits which are contained in the C164's SFRs are marked as 'Reserved'. User software should never write '1's to reserved bits. These bits are currently not implemented and may be used in future products to invoke new functions. In this case, the active state for these functions will be '1', and the inactive state will be '0'. Therefore writing only ‘0’s to reserved locations provides portability of the current software to future devices. After read accesses reserved bits should be ignored or masked out.

Serial Channels

Serial communication with other microcontrollers, processors, terminals or external peripheral components is provided by two serial interfaces with different functionality, an Asynchronous/Synchronous Serial Channel (**ASC0**) and a High-Speed Synchronous Serial Channel (**SSC**).

The ASC0 is upward compatible with the serial ports of the Siemens 8-bit microcontroller families and supports full-duplex asynchronous communication at up to 780 KBaud and half-duplex synchronous communication at up to 3.1 MBaud @ 25 MHz CPU clock.
A dedicated baud rate generator allows to set up all standard baud rates without oscillator tuning. For transmission, reception and error handling 4 separate interrupt vectors are provided. In asynchronous mode, 8- or 9-bit data frames are transmitted or received, preceded by a start bit and terminated by one or two stop bits. For multiprocessor communication, a mechanism to distinguish address from data bytes has been included (8-bit data plus wake up bit mode).
In synchronous mode, the ASC0 transmits or receives bytes (8 bits) synchronously to a shift clock which is generated by the ASC0. The ASC0 always shifts the LSB first. A loop back option is available for testing purposes.

A number of optional hardware error detection capabilities has been included to increase the reliability of data transfers. A parity bit can automatically be generated on transmission or be checked on reception. Framing error detection allows to recognize data frames with missing stop bits. An overrun error will be generated, if the last character received has not been read out of the receive buffer register at the time the reception of a new character is complete.

The SSC supports full-duplex synchronous communication at up to 6.25 Mbaud @ 25 MHz CPU clock. It may be configured so it interfaces with serially linked peripheral components. A dedicated baud rate generator allows to set up all standard baud rates without oscillator tuning. For transmission, reception and error handling 3 separate interrupt vectors are provided.
The SSC transmits or receives characters of 2...16 bits length synchronously to a shift clock which can be generated by the SSC (master mode) or by an external master (slave mode). The SSC can start shifting with the LSB or with the MSB and allows the selection of shifting and latching clock edges as well as the clock polarity.
A number of optional hardware error detection capabilities has been included to increase the reliability of data transfers. Transmit and receive error supervise the correct handling of the data buffer. Phase and baudrate error detect incorrect serial data.

The On-chip CAN Module

The integrated CAN Module handles the completely autonomous transmission and reception of CAN frames in accordance with the CAN specification V2.0 part B (active), i.e. the on-chip CAN Module can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers.

The module provides Full CAN functionality on up to 15 message objects. Message object 15 may be configured for Basic CAN functionality. Both modes provide separate masks for acceptance filtering which allows to accept a number of identifiers in Full CAN mode and also allows to disregard a number of identifiers in Basic CAN mode. All message objects can be updated independent from the other objects and are equipped for the maximum message length of 8 bytes.

The bit timing is derived from the CPU clock and is programmable up to a data rate of 1 MBaud. The CAN Module uses two pins (configurable) to interface to a bus transceiver.

A/D Converter

For analog signal measurement, a 10-bit A/D converter with 8 multiplexed input channels and a sample and hold circuit has been integrated on-chip. It uses the method of successive approximation. The sample time (for loading the capacitors) and the conversion time is programmable and can so be adjusted to the external circuitry.

Overrun error detection/protection is provided for the conversion result register (ADDAT): either an interrupt request will be generated when the result of a previous conversion has not been read from the result register at the time the next conversion is complete, or the next conversion is suspended in such a case until the previous result has been read.

For applications which require less analog input channels, the remaining channel inputs can be used as digital input port pins.

The A/D converter of the C164 supports four different conversion modes. In the standard Single Channel conversion mode, the analog level on a specified channel is sampled once and converted to a digital result. In the Single Channel Continuous mode, the analog level on a specified channel is repeatedly sampled and converted without software intervention. In the Auto Scan mode, the analog levels on a prespecified number of channels are sequentially sampled and converted. In the Auto Scan Continuous mode, the number of prespecified channels is repeatedly sampled and converted. In addition, the conversion of a specific channel can be inserted (injected) into a running sequence without disturbing this sequence. This is called Channel Injection Mode.

The Peripheral Event Controller (PEC) may be used to automatically store the conversion results into a table in memory for later evaluation, without requiring the overhead of entering and exiting interrupt routines for each data transfer.

General Purpose Timer (GPT) Unit

The GPT1 unit represents a very flexible multifunctional timer/counter structure which may be used for many different time related tasks such as event timing and counting, pulse width and duty cycle measurements, pulse generation, or pulse multiplication.

Each timer may operate independently in a number of different modes, or may be concatenated with another timer of the same module.

Each timer can be configured individually for one of four basic modes of operation, which are Timer, Gated Timer, Counter Mode and Incremental Interface Mode. In Timer Mode the input clock for a timer is derived from the internal CPU clock divided by a programmable prescaler, while Counter Mode allows a timer to be clocked in reference to external events (via TxIN).
Pulse width or duty cycle measurement is supported in Gated Timer Mode where the operation of a timer is controlled by the 'gate' level on its external input pin TxIN.
In Incremental Interface Mode the GPT1 timers can be directly connected to the incremental position sensor signals A and B via the respective inputs TxIN and TxEUD. Direction and count signals are internally derived from these two input signals, so the contents of timer Tx corresponds to the sensor position. The third position sensor signal TOP0 can be connected to an interrupt input.

The count direction (up/down) for each timer is programmable by software or may additionally be altered dynamically by an external signal (TxEUD) to facilitate e.g. position tracking.

The core timer T3 has an output toggle latch (T3OTL) which changes its state on each timer over-flow/underflow. The state of this latch may be used internally to concatenate the core timer with the respective auxiliary timers resulting in 32/33-bit timers/counters for measuring long time periods with high resolution.

Various reload or capture functions can be selected to reload timers or capture a timer's contents triggered by an external signal or a selectable transition of toggle latch T3OTL.

The maximum resolution of the timers in module GPT1 is 8 CPU clock cycles (= 16 TCL).

Capture/Compare (CAPCOM) Units

The CAPCOM units are typically used to handle high speed IO tasks such as pulse and waveform generation, pulse width modulation (PWM), Digital to Analog (D/A) conversion, software timing, or time recording relative to external events.

A number of dedicated timers with reload registers provide independent time bases for the capture/compare channels. The input clock for the timers is programmable to several prescaled values of the internal CPU clock, or may be derived from an overflow/ underflow of timer T3 in module GPT1 (for CAPCOM2 timers). This provides a wide range of variation for the timer period and resolution and allows precise adjustments to the application specific requirements. In addition, external inputs for the CAPCOM units allow event scheduling for the capture/compare registers relative to external events.

The CAPCOM2 unit supports generation and control of timing sequences on up to 16 channels (8 IO pins) with a maximum resolution of 8 CPU clock cycles. The capture/ compare register array contains 16 dual purpose capture/compare registers, each of which may be individually allocated to either CAPCOM2 timer T7 or T8, and programmed for capture or compare function. Eight registers have port pins associated with them which serve as input pins for triggering the capture function, or as output pins to indicate the occurrence of a compare event.

When a capture/compare register has been selected for capture mode, the current contents of the allocated timer will be latched (captured) into the capture/compare register in response to an external event at the port pin which is associated with this register. In addition, a specific interrupt request for this capture/compare register is generated. Either a positive, a negative, or both a positive and a negative external signal transition at the pin can be selected as the triggering event. The contents of all registers which have been selected for one of the five compare modes are continuously compared with the contents of the allocated timers. When a match occurs between the timer value and the value in a capture/compare register, specific actions will be taken based on the selected compare mode.

The CAPCOM6 unit provides 3 capture/compare channels and 1 additional compare channel. The 3 capture/compare channels can control two output lines each, which can be programmed to generate non-overlapping pulse patterns. The additional compare channel may either generate a separate output signal or modulate the output signals of the 3 other channels.

The active level for each output can be selected individually.

Versatile multichannel PWM signals can be generated, either controlled internally via a timer or externally, e.g. via hall sensors. The trap function allows to drive the outputs to a defined level in response to an external signal.

Note: Multichannel PWM modes and TRAP function are only available in devices with a full-function CAPCOM6, not in the reduced CAPCOM6.

Watchdog Timer

The Watchdog Timer represents one of the fail-safe mechanisms which have been implemented to prevent the controller from malfunctioning for longer periods of time.

The Watchdog Timer is always enabled after a reset of the chip, and can only be disabled in the time interval until the EINIT (end of initialization) instruction has been executed. Thus, the chip's start-up procedure is always monitored. The software has to be designed to service the Watchdog Timer before it overflows. If, due to hardware or software related failures, the software fails to do so, the Watchdog Timer overflows and generates an internal hardware reset and pulls the $\overline{\text{RSTOUT}}$ pin low in order to allow external hardware components to reset.

The Watchdog Timer is a 16-bit timer, clocked with the CPU clock divided by 2, 4, 128, or 256. The high byte of the Watchdog Timer register can be set to a prespecified reload value (stored in WDTREL) in order to allow further variation of the monitored time interval. Each time it is serviced by the application software, the high byte of the Watchdog Timer is reloaded. Thus, time intervals between 21 µs and 671 ms can be monitored (@ 25 MHz). The default Watchdog Timer interval after reset is 5.2 ms (@ 25 MHz).

Real Time Clock

The C164 contains a real time clock (RTC) which serves for different purposes:

- System clock to determine the current time and date,
  even during idle mode and power down mode (optionally)
- Cyclic time based interrupt, e.g. to provide a system time tick independent of the CPU frequency without loading the general purpose timers, or to wake up regularly from idle mode.
- 48-bit timer for long term measurements,
  the maximum usable timespan is more than 100 years.

The RTC module consists of a chain of 3 divider blocks, a fixed 8:1 divider, the reloadable 16-bit timer T14 and the 32-bit RTC timer (accessible via registers RTCH and RTCL). Both timers count up.

Parallel Ports

The C164 provides up to 59 IO lines which are organized into five input/output ports and one input port. All port lines are bit-addressable, and all input/output lines are individually (bit-wise) programmable as inputs or outputs via direction registers. The IO ports are true bidirectional ports which are switched to high impedance state when configured as inputs. The output drivers of three IO ports can be configured (pin by pin) for push/pull operation or open-drain operation via control registers. During the internal reset, all port pins are configured as inputs.

All port lines have programmable alternate input or output functions associated with them. PORT0 and PORT1 may be used as address and data lines when accessing external memory, while Port 4 outputs the additional segment address bits A21/19/17...A16 in systems where segmentation is used to access more than 64 KBytes of memory. Port 4 may also output the optional chip select signals $\overline{\text{CS3}}$...$\overline{\text{CS0}}$. PORT1 provides input and output signals for the CAPCOM units. Port 3 includes alternate functions of timers, serial interfaces, the optional bus control signal $\overline{\text{BHE}}$, and the system clock output (CLKOUT/FOUT). Port 5 is used for timer control signals and for the analog inputs to the A/D Converter. Port 8 provides inputs/outputs for the CAPCOM2 unit. All port lines that are not used for these alternate functions may be used as general purpose IO lines.

2.4 Power Management Features

The known basic power reduction modes (Idle and Power Down) are enhanced by a number of additional power management features (see below). These features can be combined to reduce the controller's power consumption to the respective application's possible minimum.

- Flexible clock generation
- Flexible peripheral management (peripherals can be dis/enabled separately or in groups)
- Periodic wakeup from Idle mode via RTC timer

The listed features provide effective means to realize standby conditions for the system with an optimum balance between power reduction (i.e. standby time) and peripheral operation (i.e. system functionality).

Flexible Clock Generation

The flexible clock generation system combines a variety of improved mechanisms (partly user controllable) to provide the C164 modules with clock signals. This is especially important in power sensitive modes like standby operation.

The power optimized oscillator generally reduces the amount of power which is consumed in order to generate the clock signal within the C164.

The clock system efficiently controls the amount of power which is consumed in order to distribute the clock signal within the C164.

Slowdown operation is achieved by dividing the oscillator clock by a programmable factor (1...32) resulting in a low frequency device operation which significantly reduces the overall power consumption.

Flexible Peripheral Management

The flexible peripheral management provides a mechanism to enable and disable each peripheral module separately. In each situation (e.g. several system operating modes, standby, etc.) only those peripherals may be kept running which are required for the respective functionality. All others can be switched off. It also allows the operation control of whole groups of peripherals including the power required for generating and distributing their clock input signal. Other peripherals may remain active, e.g. in order to maintain communication channels. The registers of separately disabled peripherals (not within a disabled group) can still be accessed.

Periodic Wakeup from Idle or Sleep Mode

Periodic wakeup from Idle mode or from Sleep mode combines the drastically reduced power consumption in Idle/Sleep mode (in conjunction with the additional power management features) with a high level of system availability. External signals and events can be scanned (at a lower rate) by periodically activating the CPU and selected peripherals which then return to powersave mode after a short time. This greatly reduces the system's average power consumption. Idle/Sleep mode can also be terminated by external interrupt signals.

2.5 Protected Bits

The C164 provides a special mechanism to protect bits which can be modified by the on-chip hardware from being changed unintentionally by software accesses to related bits (see also chapter “The Central Processing Unit”).

The following bits are protected:

Table 2-1 C164 Protected Bits

Register	Bit Name	Notes
T2IC, T3IC, T4IC	T2IR, T3IR, T4IR	GPT1 timer interrupt request flags
T3CON	T3OTL	GPT1 timer output toggle latches
T7IC, T8IC	T7IR, T8IR	CAPCOM2 timer interrupt request flags
S0TIC, S0TBIC	S0TIR, S0TBIR	ASC0 transmit(buffer) interrupt request flags
S0RIC, S0EIC	S0RIR, S0EIR	ASC0 receive/error interrupt request flags
S0CON	S0REN	ASC0 receiver enable flag
SSCTIC, SSCRIC	SSCTIR, SSCRIR	SSC transmit/receive interrupt request flags
SSCEIC	SSCEIR	SSC error interrupt request flag
SSCCON	SSCBSY	SSC busy flag
SSCCON	SSCBE, SSCPE	SSC error flags
SSCCON	SSCRE, SSCTE	SSC error flags
ADCIC, ADEIC	ADCIR, ADEIR	ADC end-of-conv./overrun intr. request flag
ADCON	ADST, ADCRQ	ADC start flag / injection request flag
CC31IC...CC16IC	CC31IR...CC16IR	Fast external interrupt request flags
TFR	TFR.15,14,13	Class A trap flags
TFR	TFR.7,3,2,1,0	Class B trap flags
P1H	P1H.7...P1H.4	Those bits of PORT1 used for CAPCOM2
P8	P8.3...P8.0	All bits of Port 8 used for CAPCOM2
ISNC	RTCIR	Interrupt node sharing request flag
XP0IC, XP3IC	XP0IR, XP3IR	CAN and PLL/RTC interrupt request flags

Σ = 58 protected bits.

3 Memory Organization

The memory space of the C164 is configured in a “Von Neumann” architecture. This means that code and data are accessed within the same linear address space. All of the physically separated memory areas, including internal ROM/Flash/OTP (where integrated), internal RAM, the internal Special Function Register Areas (SFRs and ESFRs), the address areas for integrated XBUS peripherals and external memory are mapped into one common address space.

The C164 provides a total addressable memory space of 16 MBytes. This address space is arranged as 256 segments of 64 KBytes each, and each segment is again subdivided into four data pages of 16 KBytes each (see figure below).

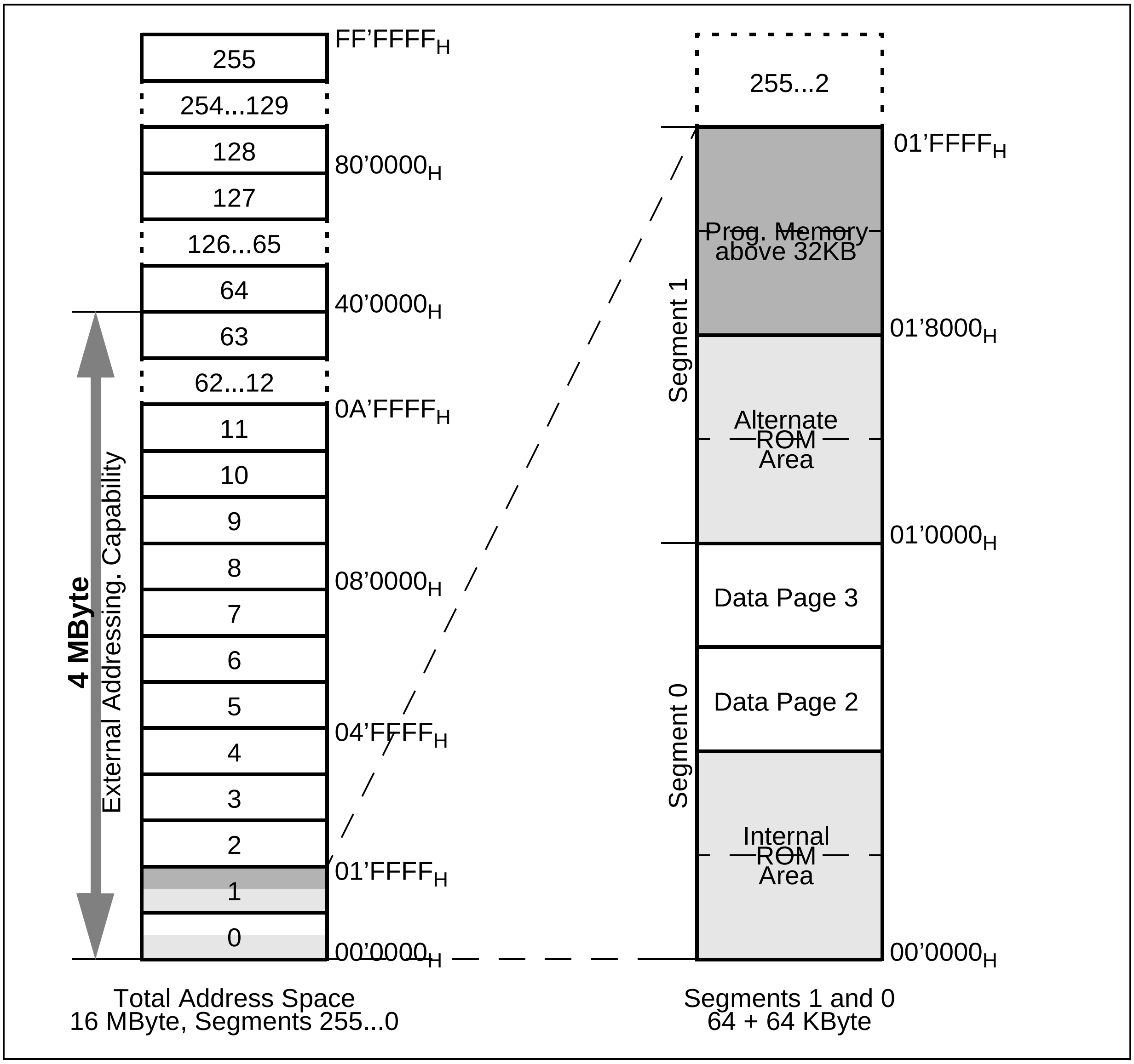


Figure 3-1 Address Space Overview

Most internal memory areas are mapped into segment 0, the system segment. The upper 4 KByte of segment 0 (00’F000$_H$...00’FFFF$_H$) hold the Internal RAM and Special Function Register Areas (SFR and ESFR). The lower 32 KByte of segment 0 (00’0000$_H$...00’7FFF$_H$) may be occupied by a part of the on-chip ROM/Flash/OTP memory and is called the Internal ROM area. This ROM area can be remapped to segment 1 (01’0000$_H$...01’7FFF$_H$), to enable external memory access in the lower half of segment 0, or the internal ROM may be disabled at all.

Code and data may be stored in any part of the internal memory areas, except for the SFR blocks, which may be used for control / data, but not for instructions.

Note: Accesses to the internal ROM area on ROMless devices will produce unpredictable results.

Bytes are stored at even or odd byte addresses. Words are stored in ascending memory locations with the low byte at an even byte address being followed by the high byte at the next odd byte address. Double words (code only) are stored in ascending memory locations as two subsequent words. Single bits are always stored in the specified bit position at a word address. Bit position 0 is the least significant bit of the byte at an even byte address, and bit position 15 is the most significant bit of the byte at the next odd byte address. Bit addressing is supported for a part of the Special Function Registers, a part of the internal RAM and for the General Purpose Registers.

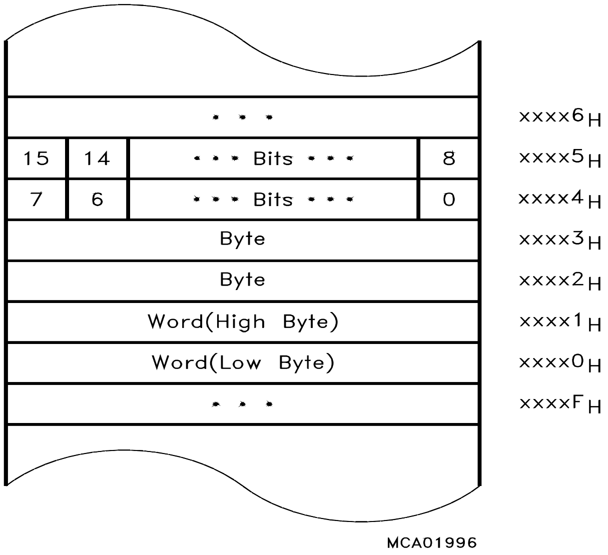


Figure 3-2 Storage of Words, Byte and Bits in a Byte Organized Memory

Note: Byte units forming a single word or a double word must always be stored within the same physical (internal, external, ROM, RAM) and organizational (page, segment) memory area.

3.1 Internal ROM Area

The C164 may reserve an address area of variable size (depending on the version) for on-chip mask-programmable ROM/Flash/OTP memory (organized as X * 32). The lower 32 KByte of this on-chip memory block are referred to as “Internal ROM Area”. Internal ROM accesses are globally enabled or disabled via bit ROMEN in register SYSCON. This bit is set during reset according to the level on pin $\overline{\text{EA}}$, or may be altered via software. If enabled, the internal ROM area occupies the lower 32 KByte of either segment 0 or segment 1 (alternate ROM area). This mapping is controlled by bit ROMS1 in register SYSCON.

Note: The size of the internal ROM area is independent of the size of the actual implemented Program Memory. Also devices with less than 32 KByte of Program Memory or with no Program Memory at all will have this 32 KByte area occupied, if the Program Memory is enabled. Devices with a larger Program Memory provide the mapping option only for the internal ROM area.

Devices with a Program Memory size above 32 KByte expand the ROM area from the middle of segment 1, i.e. starting at address 01’8000$_H$.

The internal Program Memory can be used for both code (instructions) and data (constants, tables, etc.) storage.

Code fetches are always made on even byte addresses. The highest possible code storage location in the internal Program Memory is either xx’xxFE$_H$ for single word instructions, or xx’xxFC$_H$ for double word instructions. The respective location must contain a branch instruction (unconditional), because sequential boundary crossing from internal Program Memory to external memory is not supported and causes erroneous results.

Any word and byte data read accesses may use the indirect or long 16-bit addressing modes. There is no short addressing mode for internal ROM operands. Any word data access is made to an even byte address. The highest possible word data storage location in the internal Program Memory is xx’xxFE$_H$. For PEC data transfers the internal Program Memory can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

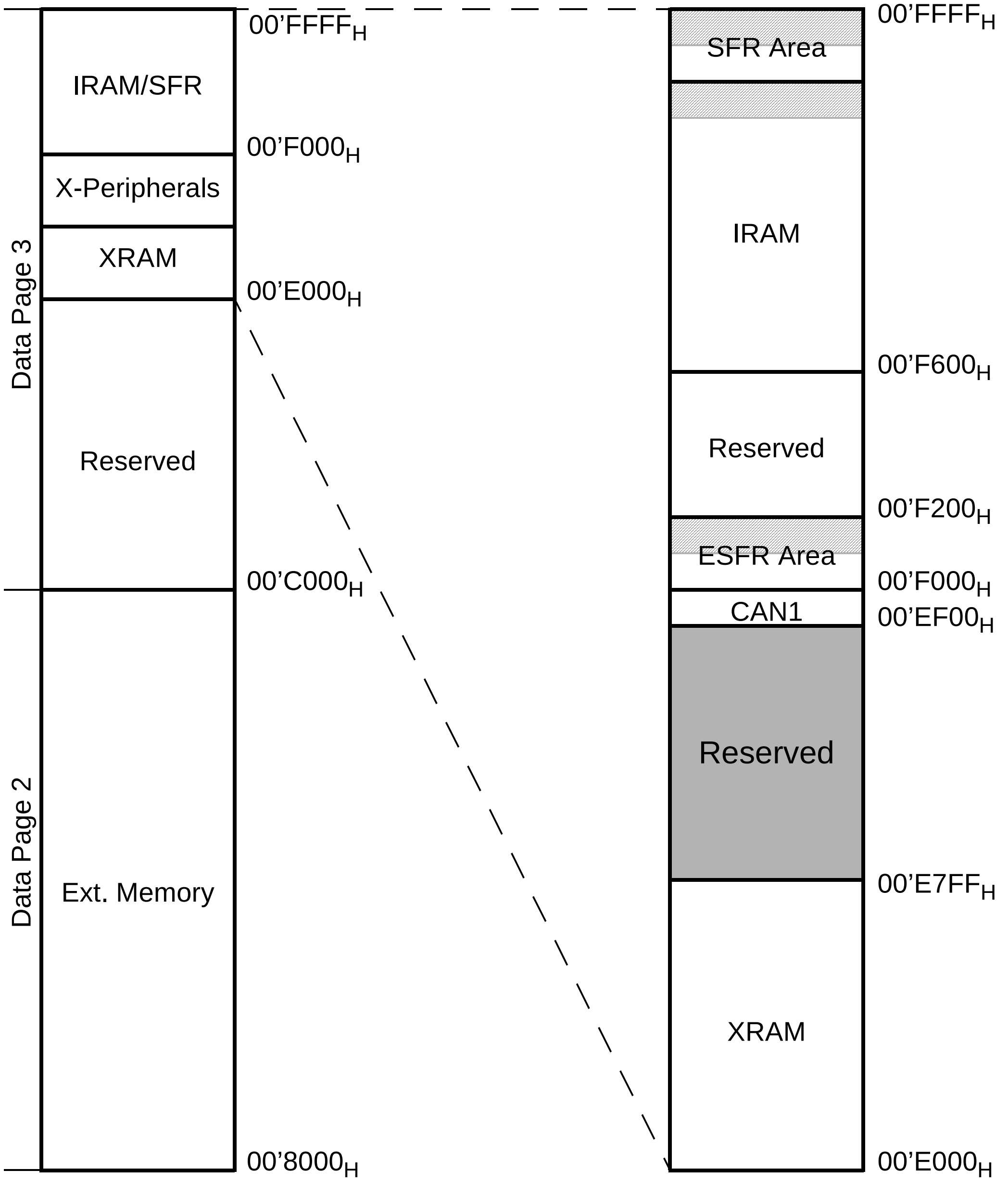
The internal Program Memory is not provided for single bit storage, and therefore it is not bit addressable.

Note: The ‘x’ in the locations above depend on the available Program Memory and on the mapping.

The internal Program Memory may be enabled, disabled or mapped into segment 0 or segment 1 under software control. Chapter “System Programming” shows how to do this and reminds of the precautions that must be taken in order to prevent the system from crashing.

3.2 Internal RAM and SFR Area

The RAM/SFR area is located within data page 3 and provides access to the internal RAM (IRAM, organized as X*16) and to two 512-Byte blocks of Special Function Registers (SFRs).
The C164 provides 2 KByte of IRAM.



Note: New XBUS peripherals will be preferably placed into the shaded areas, which now access external memory (bus cycles executed).

Figure 3-3 System Memory Map

Note: The upper 256 bytes of SFR area, ESFR area and internal RAM are bit-addressable (see hashed blocks in the figure above).

Code accesses are always made on even byte addresses. The highest possible code storage location in the internal RAM is either 00’FDFE$_H$ for single word instructions or 00’FDFC$_H$ for double word instructions. The respective location must contain a branch instruction (unconditional), because sequential boundary crossing from internal RAM to the SFR area is not supported and causes erroneous results.

Any word and byte data in the internal RAM can be accessed via indirect or long 16-bit addressing modes, if the selected DPP register points to data page 3. Any word data access is made on an even byte address. The highest possible word data storage location in the internal RAM is 00’FDFE$_H$. For PEC data transfers, the internal RAM can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

The upper 256 Byte of the internal RAM (00’FD00$_H$ through 00’FDFF$_H$) and the GPRs of the current bank are provided for single bit storage, and thus they are bit addressable.

System Stack

The system stack may be defined within the internal RAM. The size of the system stack is controlled by bitfield STKSZ in register SYSCON (see table below).

Table 3-1 System Stack Size Encoding

<STKSZ>	Stack Size (words)	Internal RAM Addresses (words)
0 0 0 $_B$	256	00’FBFE$_H$...00’FA00$_H$ (Default after Reset)
0 0 1 $_B$	128	00’FBFE$_H$...00’FB00$_H$
0 1 0 $_B$	64	00’FBFE$_H$...00’FB80$_H$
0 1 1 $_B$	32	00’FBFE$_H$...00’FBC0$_H$
1 0 0 $_B$	512	00’FBFE$_H$...00’F800$_H$
1 0 1 $_B$	---	Reserved. Do not use this combination.
1 1 0 $_B$	---	Reserved. Do not use this combination.
1 1 1 $_B$	1024	00’FDFE$_H$...00’F600$_H$ (Note: No circular stack)

For all system stack operations the on-chip RAM is accessed via the Stack Pointer (SP) register. The stack grows downward from higher towards lower RAM address locations. Only word accesses are supported to the system stack. A stack overflow (STKOV) and a stack underflow (STKUN) register are provided to control the lower and upper limits of the selected stack area. These two stack boundary registers can be used not only for protection against data destruction, but also allow to implement a circular stack with hardware supported system stack flushing and filling (except for option ’111’).
The technique of implementing this circular stack is described in chapter “System Programming”.

General Purpose Registers

The General Purpose Registers (GPRs) use a block of 16 consecutive words within the internal RAM. The Context Pointer (CP) register determines the base address of the currently active register bank. This register bank may consist of up to 16 Word-GPRs (R0, R1, ..., R15) and/or of up to 16 Byte-GPRs (RL0, RH0, ..., RL7, RH7). The sixteen Byte-GPRs are mapped onto the first eight Word-GPRs (see table below).

In contrast to the system stack, a register bank grows from lower towards higher address locations and occupies a maximum space of 32 Byte. The GPRs are accessed via short 2-, 4-, or 8-bit addressing modes using the Context Pointer (CP) register as base address (independent of the current DPP register contents). Additionally, each bit in the currently active register bank can be accessed individually.

Table 3-2 Mapping of General Purpose Registers to RAM Addresses

Internal RAM Address	Byte Registers		Word Register
<CP> + $1E_H$	---		R15
<CP> + $1C_H$	---		R14
<CP> + $1A_H$	---		R13
<CP> + 18_H	---		R12
<CP> + 16_H	---		R11
<CP> + 14_H	---		R10
<CP> + 12_H	---		R9
<CP> + 10_H	---		R8
<CP> + $0E_H$	RH7	RL7	R7
<CP> + $0C_H$	RH6	RL6	R6
<CP> + $0A_H$	RH5	RL5	R5
<CP> + 08_H	RH4	RL4	R4
<CP> + 06_H	RH3	RL3	R3
<CP> + 04_H	RH2	RL2	R2
<CP> + 02_H	RH1	RL1	R1
<CP> + 00_H	RH0	RL0	R0

The C164 supports fast register bank (context) switching. Multiple register banks can physically exist within the internal RAM at the same time. Only the register bank selected by the Context Pointer register (CP) is active at a given time, however. Selecting a new active register bank is simply done by updating the CP register. A particular Switch Context (SCXT) instruction performs register bank switching and an automatic saving of the previous context. The number of implemented register banks (arbitrary sizes) is only limited by the size of the available internal RAM.
Details on using, switching and overlapping register banks are described in chapter “System Programming”.

PEC Source and Destination Pointers

The 16 word locations in the internal RAM from 00’FCE0$_H$ to 00’FCFE$_H$ (just below the bit-addressable section) are provided as source and destination address pointers for data transfers on the eight PEC channels. Each channel uses a pair of pointers stored in two subsequent word locations with the source pointer (SRCPx) on the lower and the destination pointer (DSTPx) on the higher word address (x = 7...0).

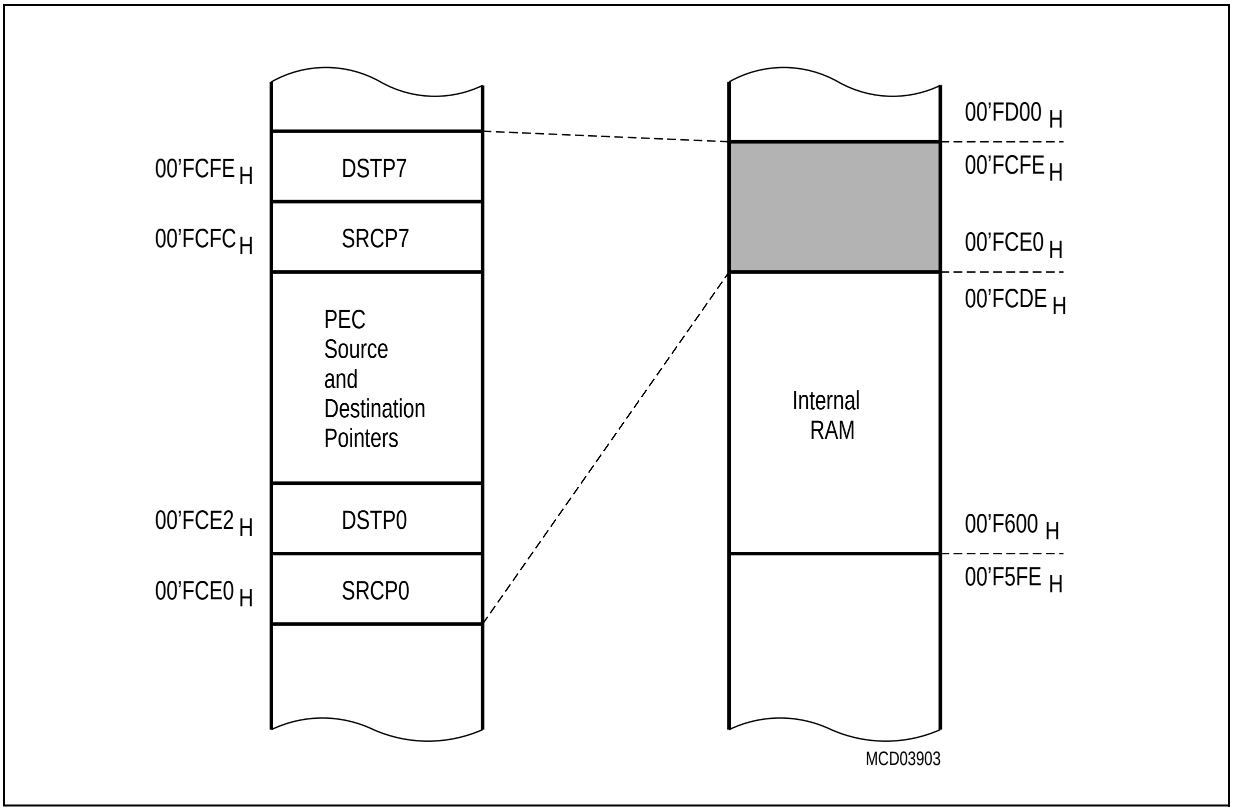


Figure 3-4 Location of the PEC Pointers

Whenever a PEC data transfer is performed, the pair of source and destination pointers, which is selected by the specified PEC channel number, is accessed independent of the current DPP register contents and also the locations referred to by these pointers are accessed independent of the current DPP register contents. If a PEC channel is not used, the corresponding pointer locations area available and can be used for word or byte data storage.
For more details about the use of the source and destination pointers for PEC data transfers see section “Interrupt and Trap Functions”.

Special Function Registers

The functions of the CPU, the bus interface, the IO ports and the on-chip peripherals of the C164 are controlled via a number of so-called Special Function Registers (SFRs). These SFRs are arranged within two areas of 512 Byte size each. The first register block, the SFR area, is located in the 512 Bytes above the internal RAM (00’FFFF$_H$...00’FE00$_H$), the second register block, the Extended SFR (ESFR) area, is located in the 512 Bytes below the internal RAM (00’F1FF$_H$...00’F000$_H$).

Special function registers can be addressed via indirect and long 16-bit addressing modes. Using an 8-bit offset together with an implicit base address allows to address word SFRs and their respective low bytes. However, this **does not work** for the respective high bytes!

Note: Writing to any byte of an SFR causes the non-addressed complementary byte to be cleared!

The upper half of each register block is bit-addressable, so the respective control/status bits can directly be modified or checked using bit addressing.

When accessing registers in the ESFR area using 8-bit addresses or direct bit addressing, an Extend Register (EXTR) instruction is required before, to switch the short addressing mechanism from the standard SFR area to the Extended SFR area. This is not required for 16-bit and indirect addresses. The GPRs R15...R0 are duplicated, i.e. they are accessible within both register blocks via short 2-, 4- or 8-bit addresses without switching.

```
ESFR_SWITCH_EXAMPLE:
EXTR    #4                  ;Switch to ESFR area for next 4 instr.
MOV     ODP2, #data16       ;ODP2 uses 8-bit reg addressing
BFLDL   DP6, #mask, #data8  ;Bit addressing for bit fields
BSET    DP1H.7              ;Bit addressing for single bits
MOV     T8REL, R1           ;T8REL uses 16-bit mem address,
                            ;R1 is duplicated into the ESFR space
                            ;(EXTR is not required for this access)
;----   ;-----------------  ;The scope of the EXTR #4 instruction...
                            ;...ends here!
MOV     T8REL, R1           ;T8REL uses 16-bit mem address,
                            ;R1 is accessed via the SFR space
```

In order to minimize the use of the EXTR instructions the ESFR area mostly holds registers which are mainly required for initialization and mode selection. Registers that need to be accessed frequently are allocated to the standard SFR area, wherever possible.

Note: The tools are equipped to monitor accesses to the ESFR area and will automatically insert EXTR instructions, or issue a warning in case of missing or excessive EXTR instructions.

3.3 The On-Chip XRAM

The C164 provides access to 2 KByte of on-chip extension RAM. The XRAM is located within data page 3 (organized as 1K*16). As the XRAM is connected to the internal XBUS it is accessed like external memory, however, no external bus cycles are executed for these accesses. XRAM accesses are globally enabled or disabled via bit XPEN in register SYSCON. This bit is cleared after reset and may be set via software during the initialization to allow accesses to the on-chip XRAM. When the XRAM is disabled (default after reset) all accesses to the XRAM area are mapped to external locations. The XRAM may be used for both code (instructions) and data (variables, user stack, tables, etc.) storage.

Code fetches are always made on even byte addresses. The highest possible code storage location in the XRAM is either 00’E7FE$_H$ for single word instructions, or 00’E7FC$_H$ for double word instructions. The respective location must contain a branch instruction (unconditional), because sequential boundary crossing from XRAM to external memory is not supported and causes erroneous results.

Any word and byte data read accesses may use the indirect or long 16-bit addressing modes. There is no short addressing mode for XRAM operands. Any word data access is made to an even byte address. The highest possible word data storage location in the XRAM is 00’E7FE$_H$. For PEC data transfers the XRAM can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

Note: As the XRAM appears like external memory it cannot be used for the C164’s system stack or register banks. The XRAM is not provided for single bit storage and therefore is not bit addressable.

The on-chip XRAM is accessed with the following bus cycles:

- Normal ALE
- No cycle time waitstates (no $\overline{\text{READY}}$ control)
- No tristate time waitstate
- No Read/Write delay
- 16-bit demultiplexed bus cycles (4 TCL).

Even if the XRAM is used like external memory it does not occupy BUSCONx/ ADDRSELx registers but rather is selected via additional dedicated XBCON/XADRS registers. These registers are mask-programmed and are not user accessible. With these registers the address area 00’E000$_H$ to 00’E7FF$_H$ is reserved for XRAM accesses.

XRAM Access via External Masters

When bit XPER-SHARE in register SYSCON is set the on-chip XRAM of the C164 can be accessed by an external master during hold mode via the C164's bus interface. These external accesses must use the same configuration as internally programmed (see above). No waitstates are required. In X-Peripheral Share mode the C164 bus interface reverses its direction, i.e. address lines (PORT1, Port 4), control signals ($\overline{\text{RD}}$, $\overline{\text{WR}}$), and $\overline{\text{BHE}}$ must be driven by the external master.

Note: The configuration in register SYSCON cannot be changed after the execution of the EINIT instruction.

3.4 External Memory Space

The C164 is capable of using an address space of up to 16 MByte. Only parts of this address space are occupied by internal memory areas. All addresses which are not used for on-chip memory (ROM/Flash/OTP or RAM) or for registers may reference external memory locations. This external memory is accessed via the C164's external bus interface.

Four memory bank sizes are supported:

- Non-segmented mode: 64 KByte with A15...A0 on PORT0 or PORT1
- 2-bit segmented mode: 256 KByte with A17...A16 on Port 4 and A15...A0 on PORT0 or PORT1
- 4-bit segmented mode: 1 MByte with A19...A16 on Port 4 and A15...A0 on PORT0 or PORT1
- 6-bit segmented mode: 4 MByte with A21...A16 on Port 4 and A15...A0 on PORT0 or PORT1

Each bank can be directly addressed via the address bus, while the programmable chip select signals can be used to select various memory banks.

The C164 also supports four **different bus types**:

- Multiplexed 16-bit Bus with address and data on PORT0 (Default after Reset)
- Multiplexed 8-bit Bus with address and data on PORT0/P0L
- Demultiplexed 16-bit Bus with address on PORT1 and data on PORT0
- Demultiplexed 8-bit Bus with address on PORT1 and data on P0L

Memory model and bus mode are selected during reset by pin $\overline{\text{EA}}$ and PORT0 pins. For further details about the external bus configuration and control please refer to chapter "The External Bus Interface".

External word and byte data can only be accessed via indirect or long 16-bit addressing modes using one of the four DPP registers. There is no short addressing mode for external operands. Any word data access is made to an even byte address.

For PEC data transfers the external memory in segment 0 can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

The external memory is not provided for single bit storage and therefore it is not bit addressable.

3.5 Crossing Memory Boundaries

The address space of the C164 is implicitly divided into equally sized blocks of different granularity and into logical memory areas. Crossing the boundaries between these blocks (code or data) or areas requires special attention to ensure that the controller executes the desired operations.

Memory Areas are partitions of the address space that represent different kinds of memory (if provided at all). These memory areas are the internal RAM/SFR area, the internal ROM/Flash/OTP (if available), the on-chip X-Peripherals (if integrated) and the external memory.

Accessing subsequent data locations that belong to different memory areas is no problem. However, when executing code, the different memory areas must be switched explicitly via branch instructions. Sequential boundary crossing is not supported and leads to erroneous results.

Note: Changing from the external memory area to the internal RAM/SFR area takes place within segment 0.

Segments are contiguous blocks of 64 KByte each. They are referenced via the code segment pointer CSP for code fetches and via an explicit segment number for data accesses overriding the standard DPP scheme.
During code fetching segments are not changed automatically, but rather must be switched explicitly. The instructions JMPS, CALLS and RETS will do this.

In larger sequential programs make sure that the highest used code location of a segment contains an unconditional branch instruction to the respective following segment, to prevent the prefetcher from trying to leave the current segment.

Data Pages are contiguous blocks of 16 KByte each. They are referenced via the data page pointers DPP3...0 and via an explicit data page number for data accesses overriding the standard DPP scheme. Each DPP register can select one of the possible 1024 data pages. The DPP register that is used for the current access is selected via the two upper bits of the 16-bit data address. Subsequent 16-bit data addresses that cross the 16 KByte data page boundaries therefore will use different data page pointers, while the physical locations need not be subsequent within memory.

3.6 Protection of the On-chip Mask ROM

The on-chip mask ROM of the C164 can be protected against read accesses of both code and data. ROM protection is established during the production process of the device (a ROM mask can be ordered with ROM protection or without it). No software control is possible, i.e. the ROM protection cannot be disabled or enabled by software.

When a device has been produced with ROM protection active, the ROM contents are protected against unauthorized access by the following measures:

- **No data read accesses** to the internal ROM by any instruction which is executed from any location outside the on-chip mask ROM (including IRAM, XRAM, and external memory).
  A program cannot read any data out of the protected ROM from outside.
  The read data will be replaced by the default value $009B_H$ for any read access to any location.
- **No codes fetches** from the internal ROM by any instruction which is executed from any location outside the on-chip mask ROM (including IRAM, XRAM, and external memory).
  A program cannot branch to a location within the protected ROM from outside. This applies to JUMPs as well as to RETurns, i.e. a called routine within RAM or external memory can never return to the protected ROM.
  The fetched code will be replaced by the default value $009B_H$ for any access to any location. This default value will be decoded as the instruction “TRAP #00” which will restart program execution at location $00'0000_H$.

Note: ROM protection may be used for applications where the complete software fits into the on-chip ROM, or where the on-chip ROM holds an initialization software which is then replaced by an external (e.g.) application software. In the latter case no data (constants, tables, etc.) can be stored within the ROM. The ROM itself should be mapped to segment 1 before branching outside, so an interrupt vector table can be established in external memory.

3.7 The On-chip Program Flash Module

The on-chip Flash module of the C164 has a capacity of 64 KByte (organized in sectors of 16 KByte, 8 KByte, 8 KByte, and 32 KByte) and combines the advantages of a very fast read access of 32 bit in one machine cycle with protected but simple writing algorithms for programming and erase. Read accesses of code and data are possible in any addressing mode, thus realizing the highest CPU performance with fetch of double word instructions in a single cycle. Based on the Flash cell concept (split gate) special algorithms for over/under-programming or erase, with verify operations, are not necessary. For optimized programming efficiency, a burst (paging) mode is offered which permits to load up to 64 Bytes into an assembly buffer with normal CPU timing before this buffer is programmed into the Flash with a store command. The algorithms for the program and erase operations are automatically controlled by the internal command state machine.

The lower 32 KBytes of the on-chip Flash memory of the C164 (sectors 0, 1, 2) can be mapped to either segment 0 (00’0000$_H$ to 00’7FFF$_H$) or segment 1 (01’0000$_H$ to 01’7FFF$_H$) during the initialization phase to allow external memory to be used for additional system flexibility. The upper 32 KBytes of the on-chip Flash memory (sector 3) are assigned to locations 01’8000$_H$ to 01’FFFF$_H$.

In standard mode (the normal operating mode) the Flash memory appears like the standard on-chip ROM of C166 Family devices with the same timing and functionality. Instruction fetches and data operand reads are performed with all addressing modes of the C166 Family instruction set.

Programming and erasing is controlled via special command sequences. This avoids inadvertent destruction of the Flash contents at a reasonably low software overhead. Command sequences consist of subsequent write (or read) accesses to virtual locations within the Flash space. These virtual locations are defined by special addresses (see command sequence table) and require register-indirect addressing. The correct execution of an operation and the general status of the Flash module can be checked via the Flash Status Register at any time.
Programming and erasing can also be controlled by an external master (e.g. a programmer) while the C164 is disabled. This possibility is described in section “External Host Mode Programming”.

Security is provided by a general read protection and a sector-specific write protection. The temporary disabling of these protection features is secured with a four-level password check sequence.

The C164 Flash module is a 1 Mbit, 5 Volt-only Flash memory organized as 16K Doublewords of 32 bit each. The physical structure of the Flash array allows simultaneous access to 64 Byte for programming operations. Programming operations take 4 ms maximum, erase operations take 64 ms maximum.

Note: Erased Flash memory cells contain all ‘0’s, contrary to standard EPROMs.

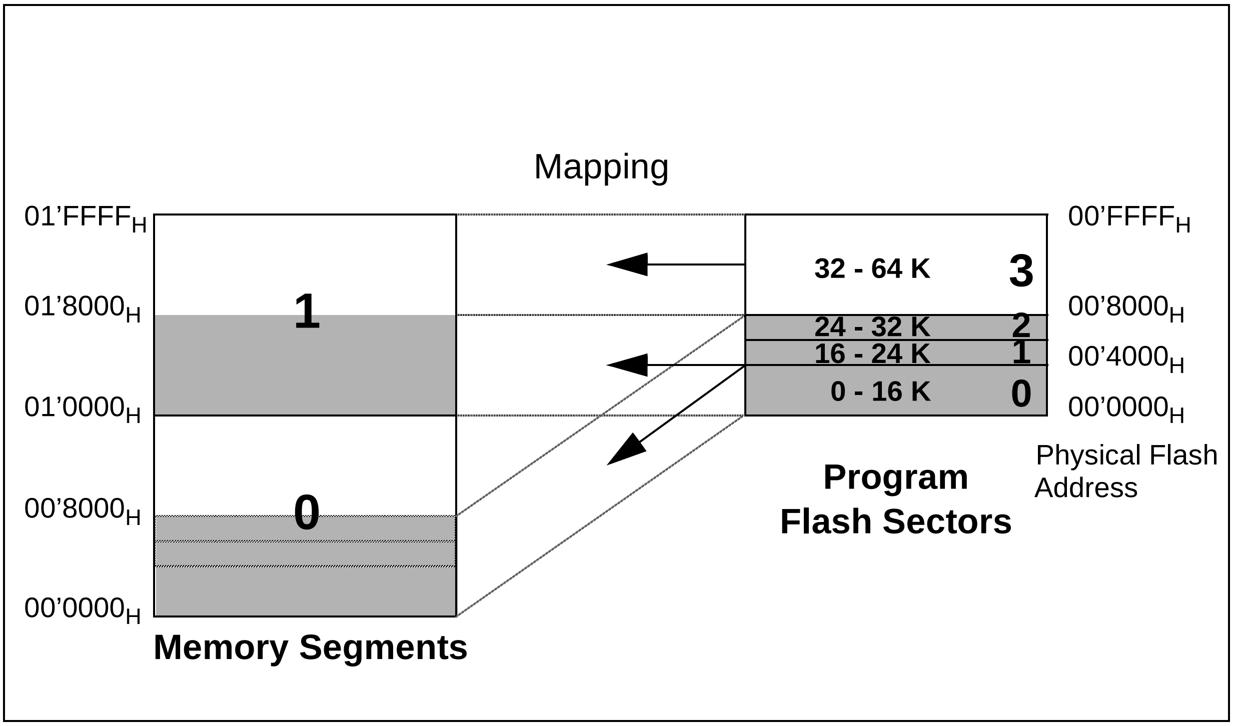


Figure 3-5 Mapping of the On-chip Flash Module Sectors

Flash Memory Configuration

Upon reset the default memory configuration of the C164 is determined by the state of its $\overline{\text{EA}}$ pin. When $\overline{\text{EA}}$ is high the startup code is fetched from the on-chip Flash memory, when $\overline{\text{EA}}$ is low the internal Flash is disabled and the startup code is fetched from external memory.

In order to access the on-chip Flash memory after booting from external memory the internal Flash must be enabled via software by setting bit ROMEN in register SYSCON. The lower 32 KBytes of the Flash memory can be mapped to segment 0 or to segment 1, controlled by bit ROMS1 in register SYSCON. Mapping to segment 1 preserves the external memory containing the startup code, while mapping to segment 0 replaces the lower 32 KBytes of the external memory with on-chip Flash memory. In this case a valid vector table must be provided in the Flash memory. As the on-chip Flash memory covers more than segment 0 segmentation should be enabled (by clearing bit SGTDIS in register SYSCON) in order to ensure correct stack handling when branching to the upper segments.
Whenever the internal memory configuration of the C164 is changed (enable, disable, mapping) the following procedure must be used to ensure correct operation:

- Configure the internal Flash as required
- Execute an inter-segment branch (JMPS, CALLS, RETS)
- Reload all four DPP registers

Note: Instructions that configure the internal Flash may only be executed from internal RAM or from external memory, ***not*** *from the Flash itself.*
Register SYSCON can only be modified ***before*** *the execution of the EINIT instruction.*

Flash Operating Modes

For the operation of the on-chip Flash module basically three operating modes can be distinguished:

In Standard Read Mode the Flash memory appears like a standard ROM allowing all code and data accesses in any addressing mode without waitstates. Standard read mode is entered:

- after the deactivation of CPU reset (max. 120 µs after reset state is finished)
- after a successful erase operation
- after a successful programming operation
- when a command sequence error is detected
- after a „reset to read“ command

Note: Standard read mode is indicated by status bit BUSY=’0’.

In Burst Mode a programming operation is prepared by writing to the Flash assembly buffer. Burst mode begins after the „Enter Burst Mode“ command sequence and ends after the „Store Burst“ command sequence. Burst mode allows the assembly (writing) of 32 words (=64 bytes) at standard CPU speed, which are then programmed in a single self-timed programming cycle.

Burst mode is only left after the „Store Burst“ command sequence, if the buffer was filled with exactly 32 words. If more or less than 32 words have been written the „Store Burst“ command will not be executed and a burst error is indicated instead (BUER=’1’).

Note: During burst mode standard read accesses can still be executed. However, the code to fill the buffer must be executed from locations outside the Flash memory (e.g. RAM or external memory).

In Command Mode the C164 executes a Flash command (erase sector, program buffer, reset state machine, etc.) which has been defined by a previous command sequence. During command mode (indicated by bit BUSY=’1’) no other Flash operations/accesses are possible except for reading the Flash status.

General rules for command sequences:

- code must be executed from locations outside the Flash memory
- all addresses must point into the active Flash space
- only register-indirect addressing is possible
- pauses between command cycles are allowed

Note: Carefully check the addresses used during command sequences. When using DPPs or EXT instructions the resulting address must be within the active Flash space. For the special addresses (see table below) bits A15...A1 are regarded. A sector address must point to the first (lowest) location within the target sector.

Detection of Problematic Bits

Flash cells store charges to represent bit levels. If the charge stored in a cell changes (e.g. due to charge coupling during operations on neighbour cells) the respective bit may be read wrong. As the charges change slowly this effect can be detected before a bit is actually read wrong. In this case also a preventive correction (via software) is possible.

A problematic bit (i.e. a bit with a changed charge) can be detected by applying a more severe comparator margin when reading a Flash location. This margin is controlled with a special command sequence (Read/Write Margin), see table.

The severe margin is selected by writing the value margin=3400$_H$ with the Read/Write Margin command sequence. A bit that returns a ’1’ when read with severe margin, while returning a ’0’ when read with standard margin, represents a problematic bit. Compare operations over a certain memory area using standard and severe margins reveal these problematic bits.

Note: Do not forget to return to standard margin by writing the value margin=0000$_H$ with the Read/Write Margin command sequence.

Problematic bits can be corrected by the application software before they lead to actual malfunctions. This is done by erasing the respective wordline (with the special erase wordline command sequence) and reprogramming it with the proper data, i.e. the data that is read when using standard margin. After each correction the corresponding sector should be verified (compared) again in order to detect problems caused by the reprogramming cycles.

Note: Wordline erase as well as margin control is only provided in order to detect problematic bits. Both features are not recommended for other usage.
Wordline erase within the same sector may only be executed twice, unless this sector is verified again, or is completely erased.

Command Sequences

The three tables below summarize the implemented command sequences for

- organizational Flash accesses,
- programming and erasing,
- protection control.

Note: Register-indirect addressing is required.

Table 3-3 Command Sequence Definitions (Organizational Accesses)

Cycle	Reset to read mode 1)	Clear status	Read flash status 2)	Read/Write Margins

1	A = 0x'AAAA$_H$
	D = xxF0$_H$
		A = 0x'AAAA$_H$
		D = xxF5$_H$
			A = 0x'AAAA$_H$
			D = xxFA$_H$
				A = 0x'AAAA$_H$
				D = xxFA$_H$

2			A = SLOC
			D = status
				A = 0x'0006$_H$
				D = margin

1) The reset to read mode command clears the four error flags and bit BUSY.

2) Between the 1st write cycle and the 2nd read cycle no instructions must be executed that use addressing mode [Rw+#data16] for the source operand (MOV, MOVB).

Notes:

SLOC is the first (lowest) location within the target sector, e.g. 01'8000$_H$ for sector 3.
margin is the control word used for margin control.
The segment part of the shown addresses (**0x**) may use any segment as long as the resulting address points to the active Flash space.

The „Read Flash status" command sequence may be executed during command mode in order to check the BUSY bit of the Flash module.

Table 3-4 Command Sequence Definitions (Programming & Erasing)

Cycle	Enter burst mode	Load burst data	Store burst buffer	Erase sector	Erase wordline
1	A = 0x’AAAA$_H$ D = xx50$_H$	A = 0x’A0F2$_H$ D = WDAT	A = 0x’AAAA$_H$ D = xxAA$_H$	A = 0x’AAAA$_H$ D = xxAA$_H$	A = 0x’AAAA$_H$ D = xxAA$_H$
2	A = WLOC D = WDAT	no read	A = 0x’5554$_H$ D = xx55$_H$	A = 0x’5554$_H$ D = xx55$_H$	A = 0x’5554$_H$ D = xx55$_H$
3	no read		A = 0x’AAAA$_H$ D = xxA0$_H$	A = 0x’AAAA$_H$ D = xx80$_H$	A = 0x’AAAA$_H$ D = xx80$_H$
4			A = WLOC D = WDAT	A = 0x’5554$_H$ D = xxAA$_H$	A = 0x’5554$_H$ D = xxAA$_H$
5				A = 0x’AAAA$_H$ D = xx55$_H$	A = 0x’AAAA$_H$ D = xx55$_H$
6				A = SLOC D = xx30$_H$	A = WLA D = xx03$_H$

Notes:

WLOC is the first (lowest) location of the 64-Byte block to which the 64-Byte buffer shall be written, e.g. 01’ABC0$_H$ or 01’AC00$_H$ (64-Byte boundary).
WDAT is the data word which shall be stored in the buffer.
SLOC is the first (lowest) location within the target sector, e.g. 01’8000$_H$ for sector 3.
WLA is the first (lowest) location of the 128-Byte wordline to be erased, e.g. 04’FF80$_H$ for the uppermost 128 Bytes (top of sector 9).
no read: during the first CPU clock cycle after the indicated command sequence the Flash module will return dummy data if a read access to the Flash area is executed. This is easily avoided if the respective next instruction does not read data from the Flash area.

The segment part of the shown addresses (**0x**) may use any segment as long as the resulting address points to the active Flash space.

The first word of programming data is written to the buffer with the „Enter burst mode“ command, the last word is written with the „Store burst buffer“ command. The medium 30 words are written with „Load burst data“ commands. Note that WLOC is the same for a complete programming sequence as the buffer address is incremented internally.

The „Read Flash status“ command sequence may be executed during command mode in order to check the BUSY bit of the Flash module.

Caution: Writing to a Flash page (space for the 64-Byte buffer) **more than once** before erasing may destroy data stored in neighbour cells! This is especially important for programming algorithms that do not write to sequential locations.

Read/Write Protection

The program flash module provides powerful and flexible protection of data and code against destruction (i.e. erasure) and undesired modification (i.e. reprogramming) as well as against undesired read access to flash contents. Two protections mechanisms can be activated:

- **Sector specific write protection** protects individual sectors against erasing and programming. This is important for the integrity of boot software and also avoids modifications of code/data by malfunction or even manipulation.
- **General read/write protection** protects the complete program flash area against all accesses from outside the module itself. This includes data read accesses as well as instruction fetches, i.e. jumps into the program flash area. The general read/write protection also disables erasing and programming of the complete program flash module.

Each protection feature is installed by user software and then remains valid permanently. Protection features may be disabled temporarily in order to reprogram portions of the flash memory or in order to call an external subroutine. Disabling and re-enabling is done under software control. However, after a reset all installed protection features are active (enabled) automatically.

All protection feature control (install, disable, re-enable) is accomplished through command sequences similar to the program/erase sequences. The two command sequences that temporarily suspend the protection feature are additionally secured by a password check sequence to ensure maximum safety against undesired accesses.

*Note: The segment part of the shown addresses (**0x**) may use any segment as long as the resulting address points to the active Flash space.*
When an "enable" command sequence (lock sector, enable read protection) is executed for the first time the respective protection feature is automatically installed permanently. For such an installation sector 0 must be unlocked (temporarily if it was locked before).
The "enable read protection" command sequence must be executed from memory outside the program flash. As this protection feature would prevent jumps into the program flash area the general read/write protection is activated only after the first instruction has been fetched from the program flash after completion of the command sequence (i.e. after a jump into the program flash).

Table 3-5 Command Sequence Definitions (Protection Control) [1]

Cycle	Lock sector	Unlock sector	Enable read protection	Disable read protection

1	A = 0x’AAAA$_H$
	D = xxAA$_H$
		A = 0x’AAAA$_H$
		D = xxAA$_H$
			A = 0x’AAAA$_H$
			D = xxAA$_H$
				A = 0x’AAAA$_H$
				D = xxAA$_H$

2	A = 0x’5554$_H$
	D = xx55$_H$
		A = 0x’5554$_H$
		D = xx55$_H$
			A = 0x’5554$_H$
			D = xx55$_H$
				A = 0x’5554$_H$
				D = xx55$_H$

3	A = 0x’AAAA$_H$
	D = xx0F$_H$
		A = 0x’AAAA$_H$
		D = xx00$_H$
			A = 0x’5E5E$_H$
			D = xx5E$_H$
				A = 0x’3C3C$_H$
				D = 00’xx3C$_H$

4	A = 0x’5554$_H$
	D = xxAA$_H$
		The 8-cycle Password Check Sequence must be executed here!	A = 0x’5554$_H$
			D = xxAA$_H$
				The 8-cycle Password Check Sequence must be executed here!

5	A = 0x’AAAA$_H$
	D = xx55$_H$
			A = 0x’AAAA$_H$
			D = xx55$_H$	

Last	A = SLOC
	D = xx0A$_H$
		A = SLOC
		D = xx05$_H$
			A = 0x’5A5A$_H$
			D = xx5A$_H$
				--- 	

Indication [2]	FSR.**SL**	FSR.**SUL**	FSR.**PROT**	FSR.**PRODI**

[1] During the first CPU clock cycle after the indicated command sequence the Flash module will return dummy data if a read access to the Flash area is executed.

[2] The Flash Status Register (FSR) provides flags that indicate the current status of the protection features.

Password Check Sequence

The command sequences that temporarily suspend protection features are critical for the overall security of the flash protection. For this reason these commands are secured by a 4-level password check sequence. For this purpose the user can store 4 arbitrary 16-bit keywords within sector 0 at the four highest locations 00’3FF8$_{H}$-00’3FFE$_{H}$ (physical or mapped to segment 0) or locations 01’3FF8$_{H}$-01’3FFE$_{H}$ (mapped to segment 1). During the password check sequence four 16-bit passwords must be entered which are internally compared with the stored keywords.

Upon a match the respective command sequence is validated and executed, i.e. the respective protection feature is suspended.

Upon a mismatch a command sequence error is indicated and the program flash modul enters the standard read mode instead.

General rules for using passwords:

- Keywords reside within sector 0 which therefore should be locked in any case to protect the keywords.
- Keywords can only be changed by erasing and re-programming sector 0.
- An aborted password check sequence (mismatch) can only be repeated after an intermediate reset.

Below is an example for the password-protected unlock sector command sequence:

```
MOV    R0, #0AAAAH                ;Auxiliary registers (R0, R1)...
MOV    R1, #05554H                ;...for special command addresses
MOV    DPP2, #000AH               ;Make R0 point to segment 2
MOV    DPP1, #0009H               ;Make R1 point to segment 2
MOV    R4, #00AAH                 ;Data for 1st command cycle
MOV    [R0], R4                   ;1st command cycle
MOV    R4, #0055H                 ;Data for 2nd command cycle
MOV    [R1], R4                   ;2nd command cycle
MOV    R4, #0000H                 ;Data for 3rd command cycle
MOV    [R0], R4                   ;3rd command cycle
MOV    DPP0, #0000H               ;DPP0:R8 = 00’3FFxH = sector 0


MOV    R8, #03FF8H                ;Location of keyword 0 (uses DPP0)
MOV    R4, #password_0
MOV    [R8], R4                   ;Write 1st password
MOV    R4, [R8]                   ;Compare with 1st keyword (R4=009BH)

MOV    R8, #03FFAH                ;Location of keyword 1 (uses DPP0)
MOV    R4, #password_1
MOV    [R8], R4                   ;Write 2nd password
MOV    R4, [R8]                   ;Compare with 2nd keyword (R4=009BH)
MOV     R8, #03FFCH                 ;Location of keyword 2 (uses DPP0)
MOV     R4, #password_2
MOV     [R8], R4                    ;Write 3rd password
MOV     R4, [R8]                    ;Compare with 3rd keyword (R4=009BH)

MOV     R8, #03FFEH                 ;Location of keyword 3 (uses DPP0)
MOV     R4, #password_3
MOV     [R8], R4                    ;Write 4th password
MOV     R4, [R8]                    ;Compare with 4th keyword (R4=009BH)

MOV     R8, #00000H                 ;Sector location (uses DPP0)
MOV     DPP0, #0006H                ;DPP0:R8 = 01'8000H = sector 3
MOV     R4, #0005H                  ;Data for last command cycle
MOV     [R8], R4                    ;Last command cycle

...                                 ;Sector 3 should now be unlocked
```

The Flash Status Register FSR reflects the overall and also sector specific status of the Flash module. Therefore the register address of FSR is the sector address SLOC (as defined above) where bits SE, SL and SUL are sector specific and all other bits are identical within each sector.

FSR
Flash Status Register (***Sector Address***) **Reset value: XXX0$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SE	**SL**	**SUL**	**-**	**PROT**	**PRODI**	**-**	**-**	**BUER**	**SQER**	**VPER**	**OPER**	**BRST**	**ERASE**	**PROG**	**BUSY**
rh	rh	rh	-	rh	rh	-	-	r(w)h	r(w)h	r(w)h	r(w)h	rh	rh	rh	rh

Bit	Function
BUSY	**Flash Busy (Summarizes the single busy bits)** 0: **Ready**. Flash command execution is completed, module is in standard read mode. 1: **Busy**. Embedded algorithms for command execution are in progress. Flash module is not in standard read mode. Cleared after reset and by “reset to read mode“ command.
PROG	**Programming State** 0: There is no programming operation in progress. 1: Flash busy with programming operation (store burst in operation). Cleared after reset.
ERASE	**Erase State** 0: There is no erase operation in progress. 1: Flash busy with erase operation. Cleared after reset.
BRST	**Burst Mode** 0: Flash not in burst mode. 1: Flash in burst mode, i.e. assembly register being filled. Burst and read mode may occur concurrently. Cleared after reset.
OPER	**Operation Error** (Cleared via „Clear status“ command) 0: Flash operation successfully terminated or currently running. 1: Flash array operation not successfully terminated, error in flash operation. Cleared by “clear status” command.

Bit	Function
VPER	**Voltage Error** (Cleared via „Clear status“ command) 0: No problem of programming voltage during Flash array operation. 1: Flash array operation not successfull because of a progamming voltage problem. Cleared by “clear status” and “reset to read mode“ command.
SQER	**Command Sequence Error** (Cleared via „Clear status“ command) 0: No command sequence error detected. 1: State machine operation aborted because of an illegal command sequence. Cleared by “clear status” and “reset to read mode“ command.
BUER	**Burst Error** (Cleared via „Clear status“ command) 0: Burst operation successfully terminated or currently running. 1: Overflow or underload condition in burst mode detected. Cleared by “clear status” and “reset to read mode“ command.
PRODI	**Protection Disabled** (Valid only if read protection is installed) 0: Flash read protection not disabled. 1: Flash read protection is temporarily disabled. Cleared after reset.
PROT	**Protected Mode** 0: Flash read protection not installed. 1: Flash read protection is permanently installed (always active after reset).
SUL	**Sector Unlocked** (Sector specific status bit) 0: Sector not unlocked (sector write protection is not disabled). 1: Sector unlocked: sector write protection is temporarily disabled. Cleared after reset.
SL	**Sector Locked** (Sector specific status bit) 0: Sector not locked. 1: Sector is permanently locked for write protection (always after reset).
SE	**Sector Erased** (Sector specific status bit) 0: Sector has not been erased since the last hardware reset. 1: Sector has been successfully erased.

Note: The status register is a read-only register. Only the four error flags in the FSR are affected with the „clear status“ command, indicated by „r(w)“. A “reset to read mode” command clears the error flags together with bit BUSY.

Operation Control and Error Handling

Command execution is started with the last command of the respective command sequence and is indicated by the respective state flag (PROG for programming, ERASE for erasing) as well as by the summarizing BUSY flag. While polling BUSY is sufficient to detect the end of a command execution it is recommended to check the error flags afterwards so an aborted command can be detected.

The command execution should therefore use the following general structure:

- Write command sequence to Flash module
- Ensure correct sequence by checking bit SQER
- Poll BUSY to determine the command termination
- Check error flags OPER, VPER, BUER (whatever is appropriate)
- If error: clear flags via „Clear status“ or „Reset“ and act upon it (e.g. with a retry operation)

The table below gives examples of software actions to be taken after a specific error has been detected:

Table 3-6 Software Reactions to Error Conditions

Detected Error	Fault Condition	Software Reaction
SQER Sequence Error	Wrong command/sector/ wordline address, wrong command code, illegal command sequence	Check address or code and repeat with correct values.
OPER Operation Error	Aborted programming or erase operation due to SW or WDT reset	Repeat Flash operation.
VPER Voltage Error	Power supply failure	Compare data. Erase sector, if data is faulty. Repeat Flash operation. Note that previous blocks must be reprogrammed after a sector erase.
BUER Burst Error	Burst buffer overrun, burst buffer underload	Repeat load sequence with correct word count.

Reset Processing

Upon a CPU reset the Flash module resets its state machine and enters the standard read mode after the internal voltages have stabilized. The internal voltages need to ramp up (e.g. after power down) or to ramp down (e.g. after an interrupted programming or erase operation). This power stabilization phase is completed after maximum 120 µs. The reset condition of CPU and Flash module is lengthened until power has stabilized.

Note: The lengthened reset condition is not reflected via pin $\overline{\text{RSTIN}}$ *in bidirectional reset mode.*
The reset lengthening is disabled in case of an external start after reset.
The delay caused by the stabilization phase must also be considered for wakeup from idle, sleep, or power down states.

3.8 The On-chip DataFlash/EEPROM Module

The on-chip DataFlash/EEPROM module of the C164 has a capacity of 4 KByte (four 1 KByte sectors) and combines the advantages of a zero waitstate read access with protected, simple but powerful writing algorithms for programming and erase. Read accesses of code and data are possible in any addressing mode. Based on the Flash cell concept (split gate) special algorithms for over/under-programming or erase, with verify operations, are not necessary.

For optimized programming efficiency the following options are provided:

- Single byte, word or page (16 Bytes / 8 Words) write access
- Single byte, word, page (16 Bytes / 8 Words) or sector erase operation
- Automatic erase-before-write operation for byte, word, page write
- Write/erase termination interrupt and error interrupt

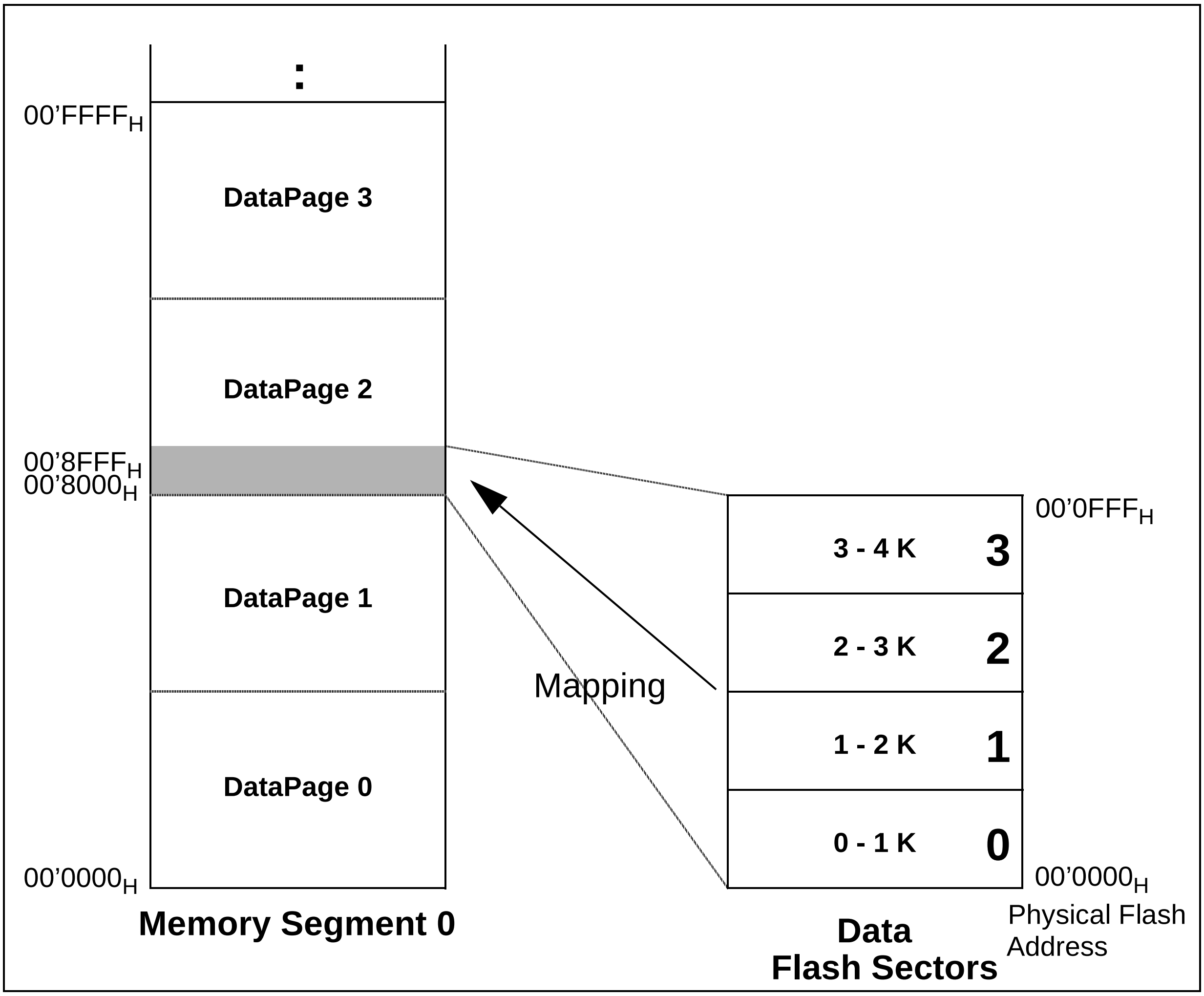


Figure 3-6 Mapping of the On-chip DataFlash/EEPROM Module Sectors

The algorithms for the program and erase operations are automatically controlled by the internal command state machine.

In standard read mode the DataFlash/EEPROM can be accessed without waitstates. Instruction fetches and data operand reads are performed with all addressing modes of the C166 Family instruction set.

Programming and erasing is controlled via special command sequences. This avoids inadvertent destruction of the Flash contents at a reasonably low software overhead. Command sequences consist of subsequent write (or read) accesses to virtual locations within the Flash space. These virtual locations are defined by special addresses (see command sequence table).

Programming and erasing can also be controlled by an external master (e.g. a programmer) while the C164 is disabled. This possibility is described in section “External Host Mode Programming”.

Sector specific write protection locks individual sectors and protects them against programming and erasing. Read accesses to locked sectors are still supported.

The C164 DataFlash/EEPROM is a 5 Volt-only Flash memory organized as 2K words of 16 bit each. The physical structure of the Flash array allows simultaneous access to 16 Byte for write operations. Programming operations take 2 ms, erase operations take 10 ms.

Note: Erased DataFlash/EEPROM cells contain all ‘0’s, contrary to standard EPROMs.

Flash Operating Modes

For the operation of the on-chip Flash module basically three operating modes can be distinguished:

In Standard Read Mode the DataFlash/EEPROM appears like a standard memory allowing all code and data accesses in any addressing mode without waitstates. Standard read mode is entered:

- after the deactivation of CPU reset (max. 120 µs after reset state is finished)
- after a successful erase operation
- after a successful programming operation
- when a command sequence error is detected
- after a „reset to read“ command

Note: Standard read mode is indicated by status bit BUSY=’0’.

In Burst Mode a programming operation is prepared by writing to the Flash assembly buffer. Burst mode begins after the „Enter Burst Mode“ command sequence and ends after the „Store Burst“ command sequence. Burst mode allows the assembly (writing) of 8 words (=16 bytes) at standard CPU speed, which are then programmed in a single self-timed programming cycle.

Burst mode should only be left with the „Store Burst“ command sequence, if the buffer was filled with exactly 8 words. If more or less than 8 words have been written the „Store Burst“ command will not produce the expected result.

Note: During burst mode standard read accesses can still be executed. However, the code to fill the buffer must be executed from locations outside the DataFlash/EEPROM (e.g. RAM or external memory).

In Command Mode the C164 executes a Flash command (erase sector, program buffer, reset state machine, etc.) which has been defined by a previous command sequence. During command mode (indicated by bit BUSY=’1’) no other Flash operations/accesses are possible except for reading the Flash status.

General rules for command sequences:

- code must be executed from locations outside the DataFlash/EEPROM
- all addresses must point into the active Flash space
- pauses between command cycles are allowed
- a pause of 2 XCLK cycles between a write access and a subsequent read access must be provided (i.e. 1 instruction cycle that does not access the flash module)

Note: Carefully check the addresses used during command sequences. When using DPPs or EXT instructions the resulting address must be within the active Flash space. For the special addresses (see table below) bits A11...A1 are regarded. A sector address must point to the first (lowest) location within the target sector.

Command Sequences

The table below summarizes the implemented command sequences and describes how to execute them:

Table 3-7 Command Sequence Definitions

Cycle	1	2	3

Reset to read mode	A = $xAAA_H$
	D = $xxF0_H$		

Clear status and interrupt flags	A = $xAAA_H$
	D = $xxF5_H$		

Read flash status [1)	A = $xAAA_H$
	D = $xxFA_H$
		A = SLOC
		D = status	

Interrupt Configuration	A = $xAAA_H$
	D = $xxF9_H$
		A = $x554_H$
		D = ICFG	

Erase byte/word	A = $xAAA_H$
	D = $xx81/2_H$
		A = WLOC
		D = $xx81/2_H$	

Write byte/word	A = $xAAA_H$
	D = $xx61/2_H$
		A = WLOC
		D = WDAT	

Erase & Write byte/word	A = $xAAA_H$
	D = $xxE1/2_H$
		A = WLOC
		D = WDAT	

Erase page	A = $xAAA_H$
	D = $xx8F_H$
		A = PLOC
		D = $xx8F_H$	

Enter burst mode	A = $xAAA_H$
	D = $xx5F_H$
		A = WLOC
		D = WDAT (first)	

Load burst data	A = $x0F2_H$
	D = WDAT		

Store burst buffer (write page)	A = $xAAA_H$
	D = $xx6F_H$
		A = WLOC
		D = WDAT (last)	

Erase & Store burst buffer	A = $xAAA_H$
	D = $xxDF_H$
		A = WLOC
		D = WDAT (last)	

Erase sector	A = $xAAA_H$
	D = $xx80_H$
		A = $x554_H$
		D = $xx55_H$
			A = SLOC
			D = $xx80_H$

Lock sector	A = $xAAA_H$
	D = $xxE0_H$
		A = $x554_H$
		D = $xx55_H$
			A = SLOC
			D = $xxE0_H$

Unlock sector	A = $xAAA_H$
	D = $xx00_H$
		A = $x554_H$
		D = $xx55_H$
			A = SLOC
			D = $xx00_H$

1) Pause of 2 XCLK cycles required between write and read access.

Notes:

WLOC is the location of the byte/word to be written or the first (lowest) location of the 16-Byte block to which the 16-Byte buffer shall be written, e.g. 00’89A0$_H$ or 00’89B0$_H$ (16-Byte boundary).
WDAT is the data word which shall be stored in the flash array or in the buffer.
PLOC is the first (lowest) location of a 16-Byte page within the DataFlash/EEPROM.
SLOC is the first (lowest) location within the target sector, e.g. 00’8C00$_H$ for sector 3.
ICFG is a control byte in which bit 0 controls the termination interrupt and bit 1 controls the error interrupt, e.g. ICFG=01$_H$ enables the termination interrupt but disables the error interrupt. After reset both interrupts are disabled. Bits 7...2 of ICFG are reserved and must be zero.

The first word of a page to be programmed is written to the buffer with the „Enter burst mode“ command, the last word is written with the „Store burst buffer“ command. The medium 6 words are written with „Load burst data“ commands. Note that WLOC is the same page location for the first and the last word as the buffer address is incremented internally.

A write byte/word command must not be issued during a burst sequence (programming of a page) as both operations require the assembly buffer and will both be aborted in such a case.

The „Read Flash status“ command sequence may be executed during command mode in order to check the BUSY bit of the Flash module.

Caution: Writing to any location within the DataFlash/EEPROM **more than once** before erasing may destroy data stored in neighbour cells! This is especially important for programming algorithms that do not write to sequential locations.

The DataFlash Status Register DFSR reflects the overall and also sector specific status of the Flash module. Therefore the register address of DFSR is the sector address SLOC (as defined above) where bits SE, SL and SUL are sector specific and all other bits are identical within each sector.

DFSR
DataFlash Status Register **(*Sector Address*)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SE	**SL**	**SUL**	**-**	**TINT**	**EINT**	**-**	**ACER**	**-**	**SQER**	**VPER**	**OPER**	**BRST**	**ERASE**	**PROG**	**BUSY**
rh	rh	rh	-	r(w)h	r(w)h	-	r(w)h	-	r(w)h	r(w)h	r(w)h	rh	rh	rh	rh

Bit	Function
BUSY	**Flash Busy** (Summarizes the single busy bits) 0: **Ready**. Flash command execution is completed, module is in standard read mode. 1: **Busy**. Embedded algorithms for command execution are in progress. Flash module is not in standard read mode.
PROG	**Programming State** 0: There is no programming operation in progress. 1: Flash busy with programming operation (store burst in operation).
ERASE	**Erase State** 0: There is no erase operation in progress. 1: Flash busy with erase operation.
BRST	**Burst Mode** 0: Flash not in burst mode. 1: Flash in burst mode, i.e. assembly register being filled. Burst and read mode may occur concurrently.
OPER	**Operation Error** (Cleared via „Clear status“ command) 0: Flash operation successfully terminated or currently running. 1: Flash array operation not successfully terminated, error in flash operation.
VPER	**Voltage Error** (Cleared via „Clear status“ command) 0: No problem of programming voltage during Flash array operation. 1: Flash array operation not successfull because of a progamming voltage problem.
SQER	**Command Sequence Error** (Cleared via „Clear status“ command) 0: No command sequence error detected. 1: State machine operation aborted because of an illegal command sequence.

Bit	Function
ACER	**Access Error** (Cleared via „Clear status“ command) 0: Read access successfully terminated. 1: Attempted read access to a busy DataFlash/EEPROM.
EINT	**Error Interrupt Flag** (Cleared via „Clear status“ command) 0: No error interrupt pending. 1: An error interrupt was generated (ACER, SQER, VPER).
TINT	**Termination Interrupt Flag** (Cleared via „Clear status“ command) 0: No termination interrupt pending. 1: An termination interrupt was generated (complete write/erase operation).
SUL	**Sector Unlocked** (Sector specific status bit) 0: Sector not unlocked (sector write protection is not disabled). 1: Sector unlocked: sector write protection is temporarily disabled.
SL	**Sector Locked** (Sector specific status bit) 0: Sector not locked. 1: Sector is permanently locked for write protection (always after reset).
SE	**Sector Erased** (Sector specific status bit) 0: Sector has not been erased since the last hardware reset. 1: Sector has been successfully erased.

Note: The status register is a read-only register. Only the four error flags and the two interrupt flags in the DFSR are affected with the „clear status“ command, indicated by „r(w)“.

DataFlash/EEPROM Address Mapping

After reset the DataFlash/EEPROM is mapped to offset 8000_H within segment 0 (see address map). By writing to the high byte of register XADRS5 an arbitrary segment within the C164's address space can be selected for the DataFlash/EEPROM.

E.g. writing 08_H to the high byte of register XADRS5 selects segment 8, i.e. the DataFlash/EEPROM starts at location $08'8000_H$.

Note: Register XADRS5 is described in section “The XBUS Interface”.
Do not remap the DataFlash/EEPROM while it delivers the instruction stream. After remapping the DataFlash/EEPROM may not be read by the next subsequent instruction (see also section “Particular Pipieline Effects”).

Operation Control and Error Handling

Command execution is started with the last command of the respective command sequence and is indicated by the respective state flag (PROG for programming, ERASE for erasing) as well as by the summarizing BUSY flag. While polling BUSY is sufficient to detect the end of a command execution it is recommended to check the error flags afterwards so an aborted command can be detected. For both a successful termination and an error condition an interrupt request can be generated.

The command execution should therefore use the following general structure:

- Write command sequence to Flash module
- Ensure correct sequence by checking bit SQER
- Poll BUSY to determine the command termination or wait for termination interrupt
- Check error flags OPER and VPER
- If error: clear flags via „Clear status“ or „Reset“ and act upon it (e.g. with a retry operation)

The table below gives examples of software actions to be taken after a specific error has been detected:

Table 3-8 Software Reactions to Error Conditions

Detected Error	Fault Condition	Software Reaction
SQER Sequence Error	Wrong command/sector address, wrong command code, illegal command sequence	Check address or code and repeat with correct values.
OPER Operation Error	Aborted programming or erase operation due to SW or WDT reset	Repeat Flash operation.
VPER Voltage Error	Power supply failure	Compare data. Erase flash area, if data is faulty. Repeat Flash operation. Note that previous blocks must be reprogrammed if a bigger area was erased (e.g. a sector).
ACER Access Error	A read access was attempted from a busy flash	Wait for non-busy state, clear status, retry operation.

Reset Processing

Upon a CPU reset the Flash module resets its state machine and enters the standard read mode after the internal voltages have stabilized. The internal voltages need to ramp up (e.g. after power down) or to ramp down (e.g. after an interrupted programming or erase operation). This power stabilization phase is completed after maximum 120 µs. During this startup time no accesses to the DataFlash/EEPROM are possible.

3.9 External Host Mode Programming

In addition to the method described above the C164's Flash memory may also programmed by external programming devices in a special mode called External Host Mode.

In External Host Mode the signals to control a programming cycle are generated by an external host using the C164's bus interface. The external host provides the command sequences and the data to be programmed (physical Flash addresses, data, and control signals). The C164 itself including the CPU is switched off and its Flash module can be accessed like standalone memory.

External Host Mode (EHM) is enabled by selecting emulation mode (P0L.0='0') and also pulling low pin P0L.5. Pins P0L.5...0 represent 01'1110$_B$ in this case.

The EHM is a variation of the emulation mode. As emulation mode is a very special operating mode, it is necessary to pay attention to the following **EHM Peculiarities**:

- Pin P0.15 (P0H.7) is inverted for the evaluation during the reset configuration.
  This influences the selected clock generation mode.
- For EHM operation direct drive or prescaler mode must be configured. If the on-chip oscillator is not supplied with a clock signal the oscillator watchdog must not be disabled, so the PLL can provide the clock signal instead.
- In emulation mode (and hence in EHM) the system clock output CLKOUT is automatically enabled. Do not drive pin CLKOUT externally.
- Signal ALE clocks the Flash state machines. If ALE transitions are not generated regularly the updating of status bits may be delayed. Do not rely on bit OPER in EHM.

The following port pins represent the interface to the C164's Flash memory in EHM:

Table 3-9 External Host Mode Interface Signals

Signal	Pin	Description
ADDR [1]	P4.1 - P4.0, P1H.7 - P1L.1	Physical Program Flash word address
DATA	P0H.7 - P0L.0	Word to be written or read
RD	RD	Read cycle control
WR	WR	Programming cycle control
CEDF	P3.10	Data Flash enable signal (EEPROM)
CEPF	P3.9	Program Flash enable signal
RSEL	P3.8	Register SELect input. Must be held low (RSEL = '0') for standard accesses to the Flash module. *Note: An active RSEL signal (RSEL = '1') could enable special test modes used for manufacturing testing (not used for operation).*
FTEST (output)	P3.7	Flash TEST signal, outputs an internal Flash clock signal of approx. 10 MHz. Can be used to verify the activation of EHM.
FBUSY (output)	P3.6	Flash BUSY signal, indicates that the Flash module is not ready for access. Active during powerdown and while $\overline{\text{CEPF}}$ = '1'.
ALE [2]	ALE	Address latch enable (indicates begin of a bus cycle and is used for synchronization) for accesses to the Program Flash
RSTOUT	RSTOUT	Generates a specific reset signal for the Flash modules, must otherwise be held high (pullup resistor)

[1] All addresses are physical (starting at $00'0000_H$) flash word addresses (address line A0 is not evaluated).

[2] For accesses to the program flash module signal ALE is used for internal synchronization and for the state machine (not evaluated for data flash module accesses). Therefore the following rules must be obeyed:
- Min. 2 dummy read cycles before the first access after reset.
- Min. 2 dummy read cycles after each command sequence.
- Min. 20 dummy read cycles after each program or erase command.
- Min. 2 dummy read cycles **before** each read status command.

The access cycles generated by the external host must fulfill the timing requirements shown in the timing diagram.

The figure below shows typical external access cycles.
Please note that $\overline{\text{CEPF}}$ must be activated at least 150 µs before the 1st access cycle.

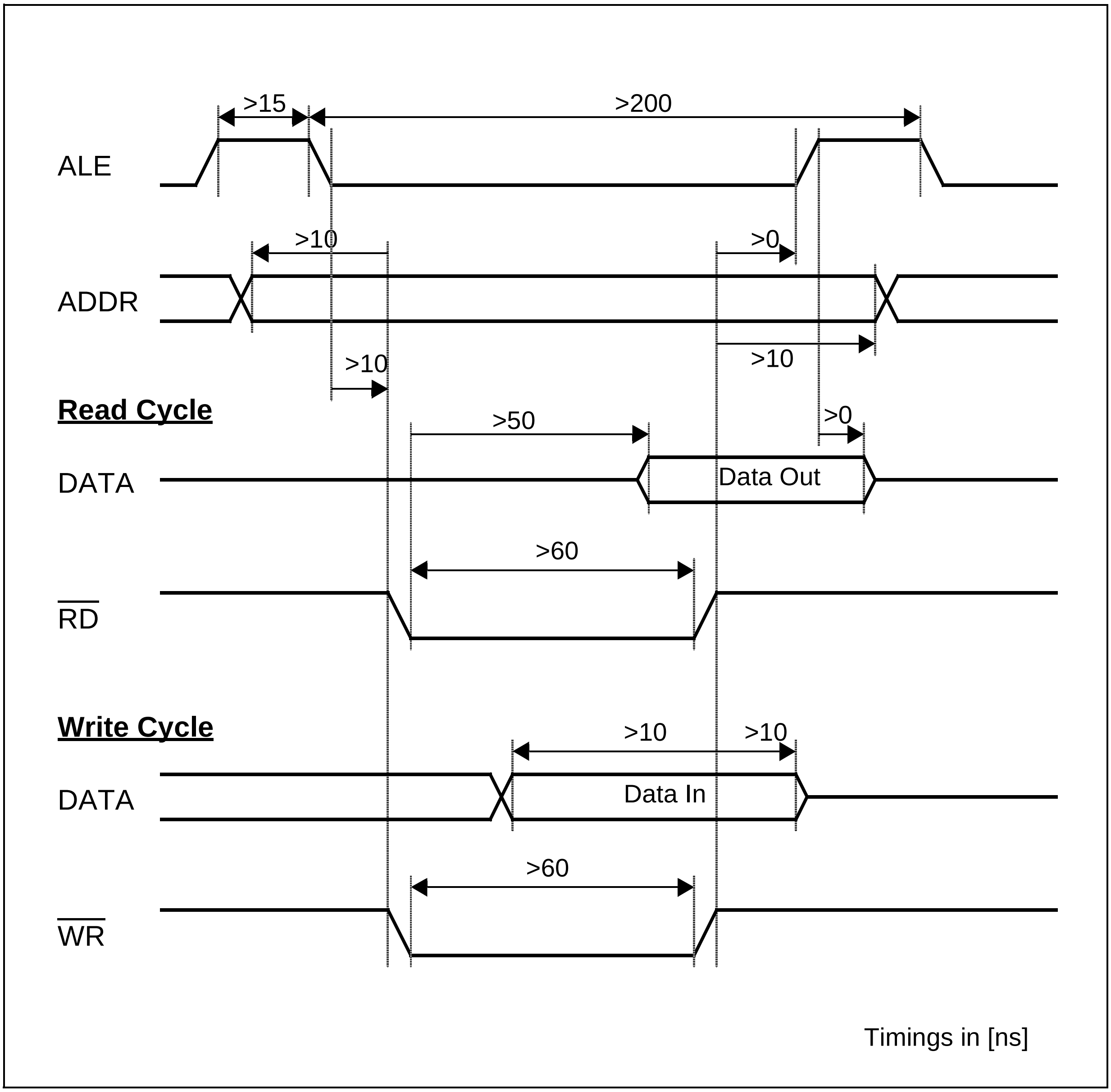


Figure 3-7 Asynchronous Flash Access Cycle in External Host Mode

4 The Central Processing Unit (CPU)

Basic tasks of the CPU are to fetch and decode instructions, to supply operands for the arithmetic and logic unit (ALU), to perform operations on these operands in the ALU, and to store the previously calculated results. As the CPU is the main engine of the C164 controller, it is also affected by certain actions of the peripheral subsystem.

Since a four stage pipeline is implemented in the C164, up to four instructions can be processed in parallel. Most instructions of the C164 are executed in one machine cycle (2 CPU clock periods) due to this parallelism.

This chapter describes how the pipeline works for sequential and branch instructions in general, and which hardware provisions have been made to speed the execution of jump instructions in particular. The general instruction timing is described including standard and exceptional timing.

While internal memory accesses are normally performed by the CPU itself, external peripheral or memory accesses are performed by a particular on-chip External Bus Controller (EBC), which is automatically invoked by the CPU whenever a code or data address refers to the external address space.

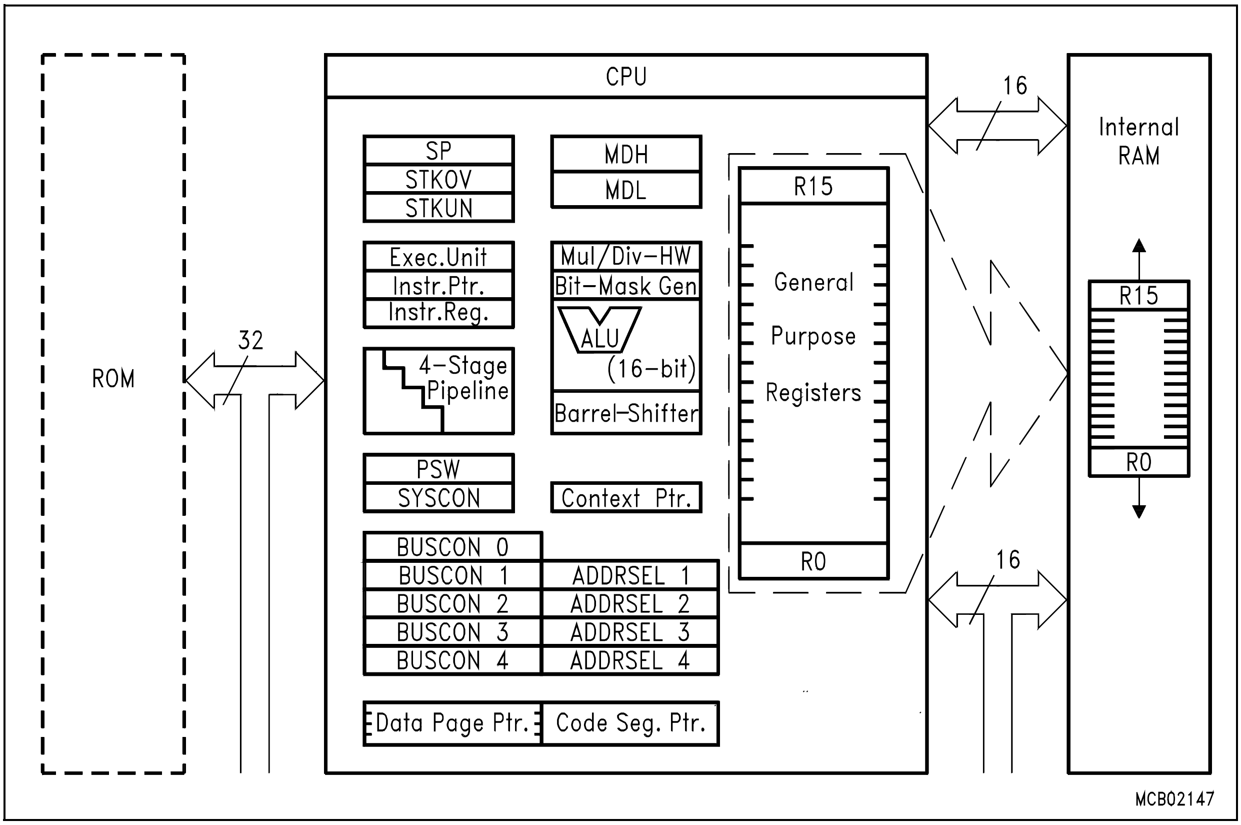


Figure 4-1 CPU Block Diagram

If possible, the CPU continues operating while an external memory access is in progress. If external data are required but are not yet available, or if a new external memory access is requested by the CPU, before a previous access has been completed, the CPU will be held by the EBC until the request can be satisfied. The EBC is described in a dedicated chapter.

The on-chip peripheral units of the C164 work nearly independent of the CPU with a separate clock generator. Data and control information is interchanged between the CPU and these peripherals via Special Function Registers (SFRs).

Whenever peripherals need a non-deterministic CPU action, an on-chip Interrupt Controller compares all pending peripheral service requests against each other and prioritizes one of them. If the priority of the current CPU operation is lower than the priority of the selected peripheral request, an interrupt will occur.

Basically, there are two types of interrupt processing:

- **Standard interrupt processing** forces the CPU to save the current program status and the return address on the stack before branching to the interrupt vector jump table.
- **PEC interrupt processing** steals just one machine cycle from the current CPU activity to perform a single data transfer via the on-chip Peripheral Event Controller (PEC).

System errors detected during program execution (socalled hardware traps) or an external non-maskable interrupt are also processed as standard interrupts with a very high priority.

In contrast to other on-chip peripherals, there is a closer conjunction between the watchdog timer and the CPU. If enabled, the watchdog timer expects to be serviced by the CPU within a programmable period of time, otherwise it will reset the chip. Thus, the watchdog timer is able to prevent the CPU from going totally astray when executing erroneous code. After reset, the watchdog timer starts counting automatically, but it can be disabled via software, if desired.

Beside its normal operation there are the following particular CPU states:

- **Reset state:** Any reset (hardware, software, watchdog) forces the CPU into a predefined active state.
- **IDLE state:** The clock signal to the CPU itself is switched off, while the clocks for the on-chip peripherals keep running.
- **POWER DOWN state:** All of the on-chip clocks are switched off (RTC clock selectable), all inputs are disregarded.
- **SLEEP state:** All of the on-chip clocks are switched off (RTC clock selectable), external interrupt inputs are enabled.

A transition into an active CPU state is forced by an interrupt (if being in IDLE or SLEEP mode) or by a reset (if being in POWER DOWN mode).
The IDLE, SLEEP, POWER DOWN, and RESET states can be entered by particular C164 system control instructions.

A set of Special Function Registers is dedicated to the functions of the CPU core:

• General System Configuration	: **SYSCON (RP0H)**
• CPU Status Indication and Control	: **PSW**
• Code Access Control	: **IP, CSP**
• Data Paging Control	: **DPP0, DPP1, DPP2, DPP3**
• GPRs Access Control	: **CP**
• System Stack Access Control	: **SP, STKUN, STKOV**
• Multiply and Divide Support	: **MDL, MDH, MDC**
• ALU Constants Support	: **ZEROS, ONES**

4.1 Instruction Pipelining

The instruction pipeline of the C164 partitiones instruction processing into four stages of which each one has its individual task:

1st –>FETCH: In this stage the instruction selected by the Instruction Pointer (IP) and the Code Segment Pointer (CSP) is fetched from either the internal ROM, internal RAM, or external memory.

2nd –>DECODE: In this stage the instructions are decoded and, if required, the operand addresses are calculated and the respective operands are fetched. For all instructions, which implicitly access the system stack, the SP register is either decremented or incremented, as specified. For branch instructions the Instruction Pointer and the Code Segment Pointer are updated with the desired branch target address (provided that the branch is taken).

3rd –>EXECUTE: In this stage an operation is performed on the previously fetched operands in the ALU. Additionally, the condition flags in the PSW register are updated as specified by the instruction. All explicit writes to the SFR memory space and all auto-increment or auto-decrement writes to GPRs used as indirect address pointers are performed during the execute stage of an instruction, too.

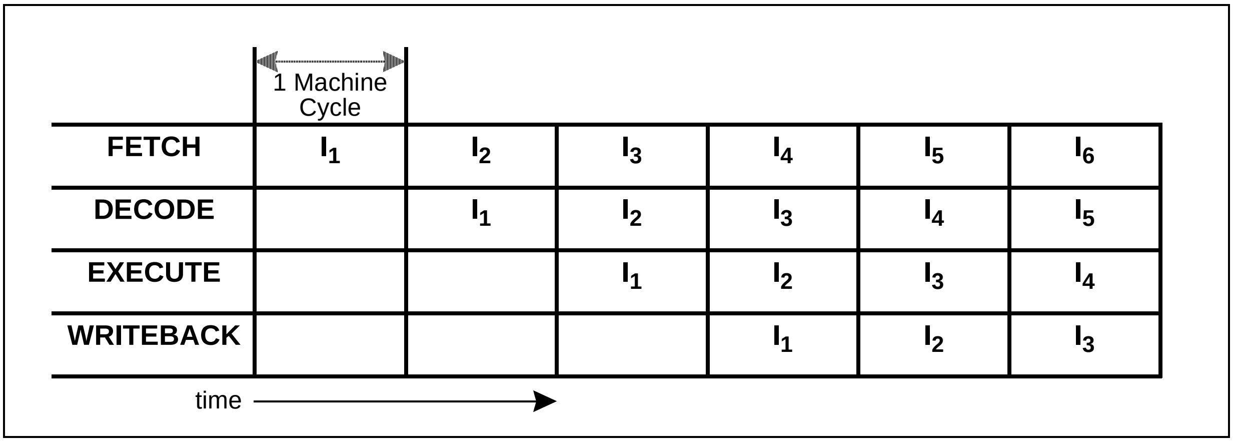
4th –>WRITE BACK: In this stage all external operands and the remaining operands within the internal RAM space are written back.

A particularity of the C164 are the so-called injected instructions. These injected instructions are generated internally by the machine to provide the time needed to process instructions, which cannot be processed within one machine cycle. They are automatically injected into the decode stage of the pipeline, and then they pass through the remaining stages like every standard instruction. Program interrupts are performed by means of injected instructions, too. Although these internally injected instructions will not be noticed in reality, they are introduced here to ease the explanation of the pipeline in the following.

Sequential Instruction Processing

Each single instruction has to pass through each of the four pipeline stages regardless of whether all possible stage operations are really performed or not. Since passing through one pipeline stage takes at least one machine cycle, any isolated instruction takes at least four machine cycles to be completed. Pipelining, however, allows parallel (i.e. simultaneous) processing of up to four instructions. Thus, most of the instructions seem to be processed during one machine cycle as soon as the pipeline has been filled once after reset (see figure below).

Instruction pipelining increases the average instruction throughput considered over a certain period of time. In the following, any execution time specification of an instruction always refers to the average execution time due to pipelined parallel instruction processing.

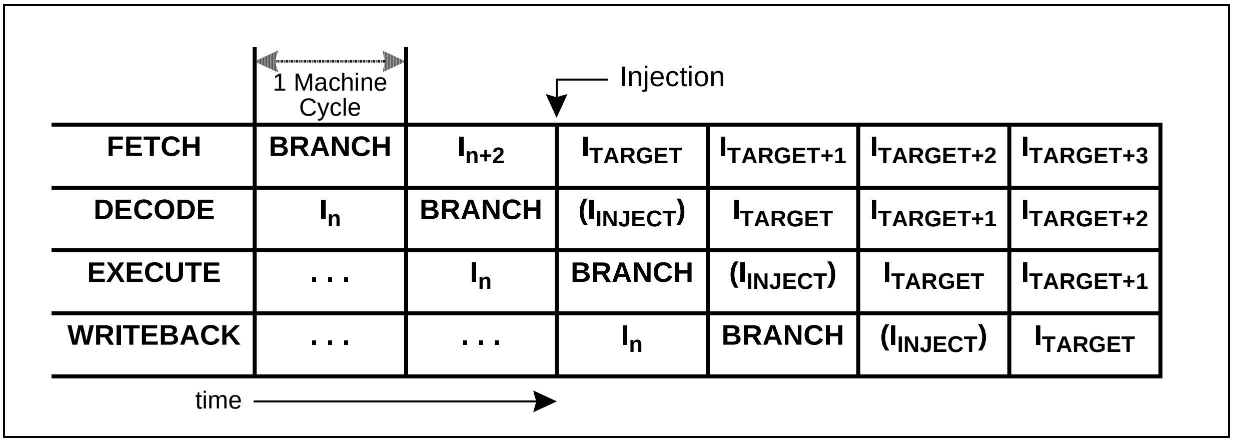


FETCH	I_1	I_2	I_3	I_4	I_5	I_6
DECODE		I_1	I_2	I_3	I_4	I_5
EXECUTE			I_1	I_2	I_3	I_4
WRITEBACK				I_1	I_2	I_3

Figure 4-2 Sequential Instruction Pipelining

Standard Branch Instruction Processing

Instruction pipelining helps to speed sequential program processing. In the case that a branch is taken, the instruction which has already been fetched providently is mostly not the instruction which must be decoded next. Thus, at least one additional machine cycle is normally required to fetch the branch target instruction. This extra machine cycle is provided by means of an injected instruction (see figure below).



FETCH	BRANCH	I_{n+2}	I_{TARGET}	$I_{TARGET+1}$	$I_{TARGET+2}$	$I_{TARGET+3}$
DECODE	I_n	BRANCH	(I_{INJECT})	I_{TARGET}	$I_{TARGET+1}$	$I_{TARGET+2}$
EXECUTE	. . .	I_n	BRANCH	(I_{INJECT})	I_{TARGET}	$I_{TARGET+1}$
WRITEBACK	. . .	. . .	I_n	BRANCH	(I_{INJECT})	I_{TARGET}

Figure 4-3 Standard Branch Instruction Pipelining

If a conditional branch is not taken, there is no deviation from the sequential program flow, and thus no extra time is required. In this case the instruction after the branch instruction will enter the decode stage of the pipeline at the beginning of the next machine cycle after decode of the conditional branch instruction.

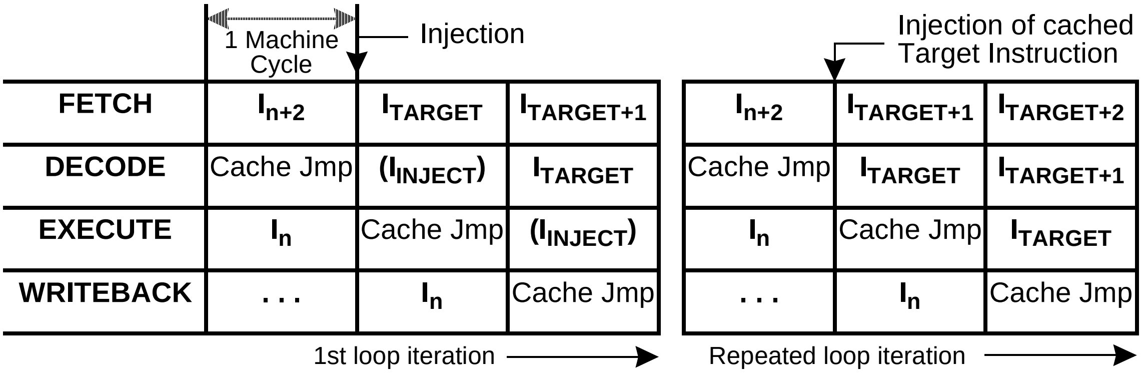
Cache Jump Instruction Processing

The C164 incorporates a jump cache to optimize conditional jumps, which are processed repeatedly within a loop. Whenever a jump on cache is taken, the extra time to fetch the branch target instruction can be saved and thus the corresponding cache jump instruction in most cases takes only one machine cycle.

This performance is achieved by the following mechanism:
Whenever a cache jump instruction passes through the decode stage of the pipeline for the first time (and provided that the jump condition is met), the jump target instruction is fetched as usual, causing a time delay of one machine cycle. In contrast to standard branch instructions, however, the target instruction of a cache jump instruction (JMPA, JMPR, JB, JBC, JNB, JNBS) is additionally stored in the cache after having been fetched.

After each repeatedly following execution of the same cache jump instruction, the jump target instruction is not fetched from progam memory but taken from the cache and immediatly injected into the decode stage of the pipeline (see figure below).

A time saving jump on cache is always taken after the second and any further occurrence of the same cache jump instruction, unless an instruction which, has the fundamental capability of changing the CSP register contents (JMPS, CALLS, RETS, TRAP, RETI), or any standard interrupt has been processed during the period of time between two following occurrences of the same cache jump instruction.



FETCH	I_{n+2}	I_{TARGET}	$I_{TARGET+1}$	I_{n+2}	$I_{TARGET+1}$	$I_{TARGET+2}$
DECODE	Cache Jmp	(I_{INJECT})	I_{TARGET}	Cache Jmp	I_{TARGET}	$I_{TARGET+1}$
EXECUTE	I_n	Cache Jmp	(I_{INJECT})	I_n	Cache Jmp	I_{TARGET}
WRITEBACK	. . .	I_n	Cache Jmp	. . .	I_n	Cache Jmp

Figure 4-4 Cache Jump Instruction Pipelining

4.2 Particular Pipeline Effects

Since up to four different instructions are processed simultaneously, additional hardware has been spent in the C164 to consider all causal dependencies which may exist on instructions in different pipeline stages without a loss of performance. This extra hardware (i.e. for 'forwarding' operand read and write values) resolves most of the possible conflicts (e.g. multiple usage of buses) in a time optimized way and thus avoids that the pipeline becomes noticeable for the user in most cases. However, there are some very rare cases, where the circumstance that the C164 is a pipelined machine requires attention by the programmer. In these cases the delays caused by pipeline conflicts can be used for other instructions in order to optimize performance.

• Context Pointer Updating

An instruction, which calculates a physical GPR operand address via the CP register, is mostly not capable of using a new CP value, which is to be updated by an immediately preceding instruction. Thus, to make sure that the new CP value is used, at least one instruction must be inserted between a CP-changing and a subsequent GPR-using instruction, as shown in the following example:

```
$I_n$     :SCXT CP,#0FC00h   ;select a new context
$I_{n+1}$ :....              ;must not be an instruction using a GPR
$I_{n+2}$ :MOV  R0,#dataX    ;write to GPR 0 in the new context
```

• Data Page Pointer Updating

An instruction, which calculates a physical operand address via a particular DPPn (n=0 to 3) register, is mostly not capable of using a new DPPn register value, which is to be updated by an immediately preceding instruction. Thus, to make sure that the new DPPn register value is used, at least one instruction must be inserted between a DPPn-changing instruction and a subsequent instruction which implicitly uses DPPn via a long or indirect addressing mode, as shown in the following example:

```
$I_n$     :MOV  DPP0,#4      ;select data page 4 via DPP0
$I_{n+1}$ :....              ;must not be an instruction using DPP0
$I_{n+2}$ :MOV  DPP0:0000H,R1;move contents of R1 to address location 01'0000$_H$
                             ;(in data page 4) supposed segment. is enabled
```

• Explicit Stack Pointer Updating

None of the RET, RETI, RETS, RETP or POP instructions is capable of correctly using a new SP register value, which is to be updated by an immediately preceding instruction. Thus, in order to use the new SP register value without erroneously performed stack accesses, at least one instruction must be inserted between an explicitly SP-writing and any subsequent of the just mentioned implicitly SP-using instructions, as shown in the following example:

```
I_n      :MOV   SP,#0FA40H    ;select a new top of stack
I_{n+1}  :....                ;must not be an instruction popping operands
                              ;from the system stack
I_{n+2}  :POP   R0            ;pop word value from new top of stack into R0
```

Note: Conflicts with instructions writing to the stack (PUSH, CALL, SCXT) are solved internally by the CPU logic.

• Controlling Interrupts

Software modifications (implicit or explicit) of the PSW are done in the execute phase of the respective instructions. In order to maintain fast interrupt responses, however, the current interrupt prioritization round does not consider these changes, i.e. an interrupt request may be acknowledged after the instruction that disables interrupts via IEN or ILVL or after the following instructions. Timecritical instruction sequences therefore should not begin directly after the instruction disabling interrupts, as shown in the following examples:

```
INTERRUPTS_OFF:
BCLR   IEN              ;globally disable interrupts
<Instr non-crit>        ;non-critical instruction
<Instr 1st-crit>        ;begin of uninterruptable critical sequence
. . .
<Instr last-crit>       ;end of uninterruptable critical sequence
INTERRUPTS_ON:
BSET   IEN              ;globally re-enable interrupts

CRITICAL_SEQUENCE:
ATOMIC #3               ;immediately block interrupts
BCLR   IEN              ;globally disable interrupts
. . .                   ;here is the uninterruptable sequence
BSET   IEN              ;globally re-enable interrupts
```

Note: The described delay of 1 instruction also applies for enabling the interrupts system i.e. no interrupt requests are acknowledged until the instruction following the enabling instruction.

• External Memory Access Sequences

The effect described here will only become noticeable, when watching the external memory access sequences on the external bus (e.g. by means of a Logic Analyzer). Different pipeline stages can simultaneously put a request on the External Bus Controller (EBC). The sequence of instructions processed by the CPU may diverge from the sequence of the corresponding external memory accesses performed by the EBC, due to the predefined priority of external memory accesses:

1st	Write Data
2nd	Fetch Code
3rd	Read Data.

• Initialization of Port Pins

Modifications of the direction of port pins (input or output) become effective only after the instruction following the modifying instruction. As bit instructions (BSET, BCLR) use internal read-modify-write sequences accessing the whole port, instructions modifying the port direction should be followed by an instruction that does not access the same port (see example below).

```
PORT_INIT_WRONG:
BSET    DP3.13                ;change direction of P3.13 to output
BSET    P3.9                  ;P3.13 is still input,
                              ;rd-mod-wr reads pin P3.13
PORT_INIT_RIGHT:
BSET    DP3.13                ;change direction of P3.13 to output
NOP                           ;any instruction not accessing port 3
BSET    P3.9                  ;P3.13 is now output,
                              ;rd-mod-wr reads P3.13’s output latch
```

• Changing the System Configuration

The instruction following an instruction that changes the system configuration via register SYSCON (e.g. the mapping of the internal ROM, segmentation, stack size) cannot use the new resources (e.g. ROM or stack). In these cases an instruction that does not access these resources should be inserted. Code accesses to the new ROM area are only possible after an absolute branch to this area.

Note: As a rule, instructions that change ROM mapping should be executed from internal RAM or external memory.

- **BUSCON/ADDRSEL**

The instruction following an instruction that changes the properties of an external address area cannot access operands within the new area. In these cases an instruction that does not access this address area should be inserted. Code accesses to the new address area should be made after an absolute branch to this area.

Note: As a rule, instructions that change external bus properties should not be executed from the respective external memory area.

- **Timing**

Instruction pipelining reduces the average instruction processing time in a wide scale (from four to one machine cycles, mostly). However, there are some rare cases, where a particular pipeline situation causes the processing time for a single instruction to be extended either by a half or by one machine cycle. Although this additional time represents only a tiny part of the total program execution time, it might be of interest to avoid these pipeline-caused time delays in time critical program modules.

Besides a general execution time description, the following section provides some hints on how to optimize time-critical program parts with regard to such pipeline-caused timing particularities.

4.3 Bit-Handling and Bit-Protection

The C164 provides several mechanisms to manipulate bits. These mechanisms either manipulate software flags within the internal RAM, control on-chip peripherals via control bits in their respective SFRs or control IO functions via port pins.

The instructions BSET, BCLR, BAND, BOR, BXOR, BMOV, BMOVN explicitly set or clear specific bits. The instructions BFLDL and BFLDH allow to manipulate up to 8 bits of a specific byte at one time. The instructions JBC and JNBS implicitly clear or set the specified bit when the jump is taken. The instructions JB and JNB (also conditional jump instructions that refer to flags) evaluate the specified bit to determine if the jump is to be taken.

Note: Bit operations on undefined bit locations will always read a bit value of '0', while the write access will not effect the respective bit location.

All instructions that manipulate single bits or bit groups internally use a read-modify-write sequence that accesses the whole word, which contains the specified bit(s).

This method has several consequences:

- Bits can only be modified within the internal address areas, i.e. internal RAM and SFRs. External locations cannot be used with bit instructions.

The upper 256 bytes of the SFR area, the ESFR area and the internal RAM are bit-addressable (see chapter "Memory Organization"), i.e. those register bits located within the respective sections can be directly manipulated using bit instructions. The other SFRs must be accessed byte/word wise.

Note: All GPRs are bit-addressable independent of the allocation of the register bank via the context pointer CP. Even GPRs which are allocated to not bit-addressable RAM locations provide this feature.

- The read-modify-write approach may be critical with hardware-effected bits. In these cases the hardware may change specific bits while the read-modify-write operation is in progress, where the writeback would overwrite the new bit value generated by the hardware. The solution is either the implemented hardware protection (see below) or realized through special programming (see "Particular Pipeline Effects").

Protected bits are not changed during the read-modify-write sequence, i.e. when hardware sets e.g. an interrupt request flag between the read and the write of the read-modify-write sequence. The hardware protection logic guarantees that only the intended bit(s) is/are effected by the write-back operation.

Note: If a conflict occurs between a bit manipulation generated by hardware and an intended software access the software access has priority and determines the final value of the respective bit.

A summary of the protected bits implemented in the C164 can be found at the end of chapter "Architectural Overview".

4.4 Instruction State Times

Basically, the time to execute an instruction depends on where the instruction is fetched from, and where possible operands are read from or written to. The fastest processing mode of the C164 is to execute a program fetched from the internal code memory. In that case most of the instructions can be processed within just one machine cycle, which is also the general minimum execution time.

All external memory accesses are performed by the C164's on-chip External Bus Controller (EBC), which works in parallel with the CPU.

This section summarizes the execution times in a very condensed way. A detailled description of the execution times for the various instructions and the specific exceptions can be found in the **"C16x Family Instruction Set Manual"**.

The table below shows the minimum execution times required to process a C164 instruction fetched from the internal code memory, the internal RAM or from external memory. These execution times apply to most of the C164 instructions - except some of the branches, the multiplication, the division and a special move instruction. In case of internal ROM program execution there is no execution time dependency on the instruction length except for some special branch situations. The numbers in the table are in units of CPU clock cycles and assume no waitstates.

Table 4-1 Minimum Execution Times

	Instruction Fetch		Word Operand Access	
Memory Area	**Word Instruction**	**Doubleword Instruction**	**Read from**	**Write to**
Internal code memory	2	2	2	---
Internal RAM	6	8	0/1	0
16-bit Demux Bus	2	4	2	2
16-bit Mux Bus	3	6	3	3
8-bit Demux Bus	4	8	4	4
8-bit Mux Bus	6	12	6	6

Execution from the internal RAM provides flexibility in terms of loadable and modifyable code on the account of execution time.
Execution from external memory strongly depends on the selected bus mode and the programming of the bus cycles (waitstates).

The operand and instruction accesses listed below can extend the execution time of an instruction:

- Internal code memory operand reads (same for byte and word operand reads)
- Internal RAM operand reads via indirect addressing modes
- Internal SFR operand reads immediately after writing
- External operand reads
- External operand writes
- Jumps to non-aligned double word instructions in the internal ROM space
- Testing Branch Conditions immediately after PSW writes

4.5 CPU Special Function Registers

The core CPU requires a set of Special Function Registers (SFRs) to maintain the system state information, to supply the ALU with register-addressable constants and to control system and bus configuration, multiply and divide ALU operations, code memory segmentation, data memory paging, and accesses to the General Purpose Registers and the System Stack.

The access mechanism for these SFRs in the CPU core is identical to the access mechanism for any other SFR. Since all SFRs can simply be controlled by means of any instruction, which is capable of addressing the SFR memory space, a lot of flexibility has been gained, without the need to create a set of system-specific instructions.

Note, however, that there are user access restrictions for some of the CPU core SFRs to ensure proper processor operations. The instruction pointer IP and code segment pointer CSP cannot be accessed directly at all. They can only be changed indirectly via branch instructions.

The PSW, SP, and MDC registers can be modified not only explicitly by the programmer, but also implicitly by the CPU during normal instruction processing. Note that any explicit write request (via software) to an SFR supersedes a simultaneous modification by hardware of the same register.

Note: Any write operation to a single byte of an SFR clears the non-addressed complementary byte within the specified SFR.
Non-implemented (reserved) SFR bits cannot be modified, and will always supply a read value of ’0’.

The System Configuration Register SYSCON

This bit-addressable register provides general system configuration and control functions. The reset value for register SYSCON depends on the state of the PORT0 pins during reset (see hardware effectable bits).

SYSCON
System Control Register **SFR ($FF12_H/89_H$)** **Reset value: $0XX0_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STKSZ			**ROM S1**	**SGT DIS**	**ROM EN**	**BYT DIS**	**CLK EN**	**WR CFG**	**CS CFG**	**-**	**OWD DIS**	**BD RST EN**	**XPEN**	**VISI-BLE**	**-**
rw			rw	rw	rwh	rwh	rw	rwh	rw	-	rwh	rw	rw	rw	-

Bit	Function
VISIBLE	**Visible Mode Control** 0: Accesses to XBUS peripherals are done internally 1: XBUS peripheral accesses are made visible on the external pins
XPEN	**XBUS Peripheral Enable Bit** 0: Accesses to the on-chip X-Peripherals and their functions are disabled 1: The on-chip X-Peripherals are enabled and can be accessed
BDRSTEN	**Bidirectional Reset Enable Bit** 0: Pin $\overline{\text{RSTIN}}$ is an input only. 1: Pin $\overline{\text{RSTIN}}$ is pulled low during the internal reset sequence after any reset.
OWDDIS	**Oscillator Watchdog Disable Bit** 0: The on-chip oscillator watchdog is enabled and active. 1: The on-chip oscillator watchdog is disabled and the CPU clock is always fed from the oscillator input.
CSCFG	**Chip Select Configuration Control** 0: Latched $\overline{\text{CS}}$ mode. The $\overline{\text{CS}}$ signals are latched internally and driven to the (enabled) port pins synchronously. 1: Unlatched $\overline{\text{CS}}$ mode. The $\overline{\text{CS}}$ signals are directly derived from the address and driven to the (enabled) port pins.
WRCFG	**Write Configuration Control** (Set according to pin P0H.0 during reset) 0: Pins $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ retain their normal function 1: Pin $\overline{\text{WR}}$ acts as $\overline{\text{WRL}}$, pin $\overline{\text{BHE}}$ acts as $\overline{\text{WRH}}$
CLKEN	**System Clock Output Enable** (CLKOUT) 0: CLKOUT disabled: pin may be used for general purpose IO or for signal FOUT 1: CLKOUT enabled: pin outputs the system clock signal

Bit	Function
BYTDIS	**Disable/Enable Control for Pin $\overline{\text{BHE}}$** (Set according to data bus width) 0: Pin $\overline{\text{BHE}}$ enabled 1: Pin $\overline{\text{BHE}}$ disabled, pin may be used for general purpose IO
ROMEN	**Internal ROM Enable** (Set according to pin $\overline{\text{EA}}$ during reset) 0: Internal program memory disabled, accesses to the ROM area use the external bus 1: Internal program memory enabled
SGTDIS	**Segmentation Disable/Enable Control** 0: Segmentation enabled (CSP is saved/restored during interrupt entry/exit) 1: Segmentation disabled (Only IP is saved/restored)
ROMS1	**Internal ROM Mapping** 0: Internal ROM area mapped to segment 0 ($00'0000_H$...$00'7FFF_H$) 1: Internal ROM area mapped to segment 1 ($01'0000_H$...$01'7FFF_H$)
STKSZ	**System Stack Size** Selects the size of the system stack (in the internal RAM) from 32 to 512 words

Note: Register SYSCON cannot be changed after execution of the EINIT instruction. The function of bits VISIBLE, WRCFG, BYTDIS, ROMEN and ROMS1 is described in more detail in chapter “The External Bus Controller”.

System Clock Output Enable (CLKEN)

The system clock output function is enabled by setting bit CLKEN in register SYSCON to ’1’. If enabled, port pin P3.15 takes on its alternate function as CLKOUT output pin. The clock output is a 50 % duty cycle clock (except for direct drive operation where CLKOUT reflects the clock input signal, and for slowdown operation where CLKOUT mirrors the CPU clock signal) whose frequency equals the CPU operating frequency ($f_{OUT} = f_{CPU}$).

Note: The output driver of port pin P3.15 is switched on automatically, when the CLKOUT function is enabled. The port direction bit is disregarded.
After reset, the clock output function is disabled (CLKEN = ‘0’).
In emulation mode the CLKOUT function is enabled automatically.

Segmentation Disable/Enable Control (SGTDIS)

Bit SGTDIS allows to select either the segmented or non-segmented memory mode.
In non-segmented memory mode (SGTDIS=’1’) it is assumed that the code address space is restricted to 64 KBytes (segment 0) and thus 16 bits are sufficient to represent all code addresses. For implicit stack operations (CALL or RET) the CSP register is totally ignored and only the IP is saved to and restored from the stack.
In segmented memory mode (SGTDIS=’0’) it is assumed that the whole address space is available for instructions. For implicit stack operations (CALL or RET) the CSP register and the IP are saved to and restored from the stack. After reset the segmented memory mode is selected.

Note: Bit SGTDIS controls if the CSP register is pushed onto the system stack in addition to the IP register before an interrupt service routine is entered, and it is repopped when the interrupt service routine is left again.

System Stack Size (STKSZ)

This bitfield defines the size of the physical system stack, which is located in the internal RAM of the C164. An area of 32...512 words or all of the internal RAM may be dedicated to the system stack. A so-called “circular stack” mechanism allows to use a bigger virtual stack than this dedicated RAM area.

These techniques as well as the encoding of bitfield STKSZ are described in more detail in chapter “System Programming”.

The Processor Status Word PSW

This bit-addressable register reflects the current state of the microcontroller. Two groups of bits represent the current ALU status, and the current CPU interrupt status. A separate bit (USR0) within register PSW is provided as a general purpose user flag.

PSW

Program Status Word **SFR ($FF10_H$/88_H)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILVL				**IEN**	**-**	**-**	**-**	**-**	**USR0**	**MULIP**	**E**	**Z**	**V**	**C**	**N**
rwh				rw	-	-	-	-	rw	rwh	rwh	rwh	rwh	rwh	rwh

Bit	Function
N	**Negative Result** Set, when the result of an ALU operation is negative.
C	**Carry Flag** Set, when the result of an ALU operation produces a carry bit.
V	**Overflow Result** Set, when the result of an ALU operation produces an overflow.
Z	**Zero Flag** Set, when the result of an ALU operation is zero.
E	**End of Table Flag** Set, when the source operand of an instruction is 8000_H or 80_H.
MULIP	**Multiplication/Division In Progress** 0: There is no multiplication/division in progress. 1: A multiplication/division has been interrupted.
USR0	**User General Purpose Flag** May be used by the application software.
ILVL, IEN	**Interrupt and EBC Control Fields** Define the response to interrupt requests. (Described in section “Interrupt and Trap Functions”)

ALU Status (N, C, V, Z, E, MULIP)

The condition flags (N, C, V, Z, E) within the PSW indicate the ALU status due to the last recently performed ALU operation. They are set by most of the instructions due to specific rules, which depend on the ALU or data movement operation performed by an instruction.

After execution of an instruction which explicitly updates the PSW register, the condition flags cannot be interpreted as described in the following, because any explicit write to the PSW register supersedes the condition flag values, which are implicitly generated by the CPU. Explicitly reading the PSW register supplies a read value which represents the state of the PSW register after execution of the immediately preceding instruction.

Note: After reset, all of the ALU status bits are cleared.

- **N-Flag:** For most of the ALU operations, the N-flag is set to ’1’, if the most significant bit of the result contains a ’1’, otherwise it is cleared. In the case of integer operations the N-flag can be interpreted as the sign bit of the result (negative: N=’1’, positive: N=’0’). Negative numbers are always represented as the 2's complement of the corresponding positive number. The range of signed numbers extends from '–8000_H' to '+$7FFF_H$' for the word data type, or from '–80_H' to '+$7F_H$' for the byte data type.For Boolean bit operations with only one operand the N-flag represents the previous state of the specified bit. For Boolean bit operations with two operands the N-flag represents the logical XORing of the two specified bits.

- **C-Flag:** After an addition the C-flag indicates that a carry from the most significant bit of the specified word or byte data type has been generated. After a subtraction or a comparison the C-flag indicates a borrow, which represents the logical negation of a carry for the addition.
This means that the C-flag is set to ’1’, if **no** carry from the most significant bit of the specified word or byte data type has been generated during a subtraction, which is performed internally by the ALU as a 2’s complement addition, and the C-flag is cleared when this complement addition caused a carry.
The C-flag is always cleared for logical, multiply and divide ALU operations, because these operations cannot cause a carry anyhow.
For shift and rotate operations the C-flag represents the value of the bit shifted out last. If a shift count of zero is specified, the C-flag will be cleared. The C-flag is also cleared for a prioritize ALU operation, because a ’1’ is never shifted out of the MSB during the normalization of an operand.
For Boolean bit operations with only one operand the C-flag is always cleared. For Boolean bit operations with two operands the C-flag represents the logical ANDing of the two specified bits.

- **V-Flag:** For addition, subtraction and 2’s complementation the V-flag is always set to ’1’, if the result overflows the maximum range of signed numbers, which are representable by either 16 bits for word operations ('–8000_H' to '+$7FFF_H$'), or by 8 bits for byte operations ('–80_H' to '+$7F_H$'), otherwise the V-flag is cleared. Note that the result of an integer addition, integer subtraction, or 2's complement is not valid, if the V-flag indicates an arithmetic overflow.

For multiplication and division the V-flag is set to ’1’, if the result cannot be represented in a word data type, otherwise it is cleared. Note that a division by zero will always cause an overflow. In contrast to the result of a division, the result of a multiplication is valid regardless of whether the V-flag is set to ’1’ or not.
Since logical ALU operations cannot produce an invalid result, the V-flag is cleared by these operations.

The V-flag is also used as ’Sticky Bit’ for rotate right and shift right operations. With only using the C-flag, a rounding error caused by a shift right operation can be estimated up to a quantity of one half of the LSB of the result. In conjunction with the V-flag, the C-flag allows evaluating the rounding error with a finer resolution (see table below).
For Boolean bit operations with only one operand the V-flag is always cleared. For Boolean bit operations with two operands the V-flag represents the logical ORing of the two specified bits.

Table 4-2 Shift Right Rounding Error Evaluation

C-Flag	V-Flag	Rounding Error Quantity		
0	0	-	No rounding error	-
0	1	0 <	Rounding error	< $^1/_2$ LSB
1	0		Rounding error	= $^1/_2$ LSB
1	1		Rounding error	> $^1/_2$ LSB

- **Z-Flag:** The Z-flag is normally set to ’1’, if the result of an ALU operation equals zero, otherwise it is cleared.
For the addition and subtraction with carry the Z-flag is only set to ’1’, if the Z-flag already contains a ’1’ and the result of the current ALU operation additionally equals zero. This mechanism is provided for the support of multiple precision calculations.
For Boolean bit operations with only one operand the Z-flag represents the logical negation of the previous state of the specified bit. For Boolean bit operations with two operands the Z-flag represents the logical NORing of the two specified bits. For the prioritize ALU operation the Z-flag indicates, if the second operand was zero or not.

- **E-Flag:** The E-flag can be altered by instructions, which perform ALU or data movement operations. The E-flag is cleared by those instructions which cannot be reasonably used for table search operations. In all other cases the E-flag is set depending on the value of the source operand to signify whether the end of a search table is reached or not. If the value of the source operand of an instruction equals the lowest negative number, which is representable by the data format of the corresponding instruction (’8000_H’ for the word data type, or ’80_H’ for the byte data type), the E-flag is set to ’1’, otherwise it is cleared.

- **MULIP-Flag:** The MULIP-flag will be set to ’1’ by hardware upon the entrance into an interrupt service routine, when a multiply or divide ALU operation was interrupted before completion. Depending on the state of the MULIP bit, the hardware decides whether a multiplication or division must be continued or not after the end of an interrupt service. The MULIP bit is overwritten with the contents of the stacked MULIP-flag when the return-from-interrupt-instruction (RETI) is executed. This normally means that the MULIP-flag is cleared again after that.

*Note: The MULIP flag is a part of the task environment! When the interrupting service routine does not return to the interrupted multiply/divide instruction (*i.e. *in case of a task scheduler that switches between independent tasks), the MULIP flag must be saved as part of the task environment and must be updated accordingly for the new task before this task is entered.*

CPU Interrupt Status (IEN, ILVL)

The Interrupt Enable bit allows to globally enable (IEN=’1’) or disable (IEN=’0’) interrupts. The four-bit Interrupt Level field (ILVL) specifies the priority of the current CPU activity. The interrupt level is updated by hardware upon entry into an interrupt service routine, but it can also be modified via software to prevent other interrupts from being acknowledged. In case an interrupt level '15' has been assigned to the CPU, it has the highest possible priority, and thus the current CPU operation cannot be interrupted except by hardware traps or external non-maskable interrupts. For details please refer to chapter “Interrupt and Trap Functions”.

After reset all interrupts are globally disabled, and the lowest priority (ILVL=0) is assigned to the initial CPU activity.

The Instruction Pointer IP

This register determines the 16-bit intra-segment address of the currently fetched instruction within the code segment selected by the CSP register. The IP register is not mapped into the C164's address space, and thus it is not directly accessable by the programmer. The IP can, however, be modified indirectly via the stack by means of a return instruction.

The IP register is implicitly updated by the CPU for branch instructions and after instruction fetch operations.

IP
Instruction Pointer **- - - (- - - -/- -)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ip															
(r)(w)h															

Bit	Function
ip	Specifies the intra segment offset, from where the current instruction is to be fetched. IP refers to the current segment <SEGNR>.

The Code Segment Pointer CSP

This non-bit addressable register selects the code segment being used at run-time to access instructions. The lower 8 bits of register CSP select one of up to 256 segments of 64 KBytes each, while the upper 8 bits are reserved for future use.

CSP
Code Segment Pointer **SFR ($FE08_H$/04_H)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	**SEGNR**							
-	-	-	-	-	-	-	-	rh							

Bit	Function
SEGNR	**Segment Number** Specifies the code segment, from where the current instruction is to be fetched. SEGNR is ignored, when segmentation is disabled.

Code memory addresses are generated by directly extending the 16-bit contents of the IP register by the contents of the CSP register as shown in the figure below.

In case of the segmented memory mode the selected number of segment address bits (via bitfield SALSEL) of register CSP is output on the respective segment address pins of Port 4 for all external code accesses. For non-segmented memory mode or Single Chip Mode the content of this register is not significant, because all code acccesses are automatically restricted to segment 0.

Note: The CSP register can only be read but not written by data operations. It is, however, modified either directly by means of the JMPS and CALLS instructions, or indirectly via the stack by means of the RETS and RETI instructions.
Upon the acceptance of an interrupt or the execution of a software TRAP instruction, the CSP register is automatically set to zero.

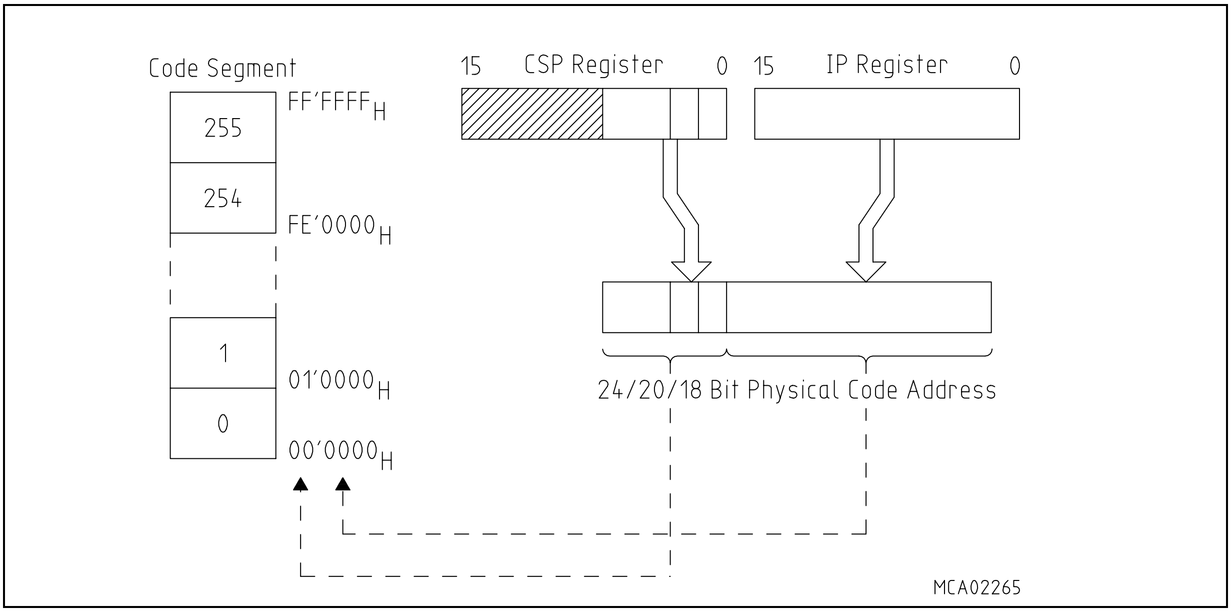


Figure 4-5 Addressing via the Code Segment Pointer

Note: When segmentation is disabled, the IP value is used directly as the 16-bit address.

The Data Page Pointers DPP0, DPP1, DPP2, DPP3

These four non-bit addressable registers select up to four different data pages being active simultaneously at run-time. The lower 10 bits of each DPP register select one of the 1024 possible 16-Kbyte data pages while the upper 6 bits are reserved for future use. The DPP registers allow to access the entire memory space in pages of 16 Kbytes each.

DPP0
Data Page Pointer 0 **SFR ($FE00_H/00_H$)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	DPP0PN									
-	-	-	-	-	-	rw									

DPP1
Data Page Pointer 1 **SFR ($FE02_H/01_H$)** **Reset value: 0001_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	DPP1PN									
-	-	-	-	-	-	rw									

DPP2
Data Page Pointer 2 **SFR ($FE04_H/02_H$)** **Reset value: 0002_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	DPP2PN									
-	-	-	-	-	-	rw									

DPP3
Data Page Pointer 3 **SFR ($FE06_H/03_H$)** **Reset value: 0003_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	DPP3PN									
-	-	-	-	-	-	rw									

Bit	Function
DPPxPN	**Data Page Number of DPPx** Specifies the data page selected via DPPx. Only the least significant two bits of DPPx are significant, when segmentation is disabled.

The DPP registers are implicitly used, whenever data accesses to any memory location are made via indirect or direct long 16-bit addressing modes (except for override accesses via EXTended instructions and PEC data transfers). After reset, the Data Page Pointers are initialized in a way that all indirect or direct long 16-bit addresses result in identical 18-bit addresses. This allows to access data pages 3...0 within segment 0 as shown in the figure below. If the user does not want to use any data paging, no further action is required.

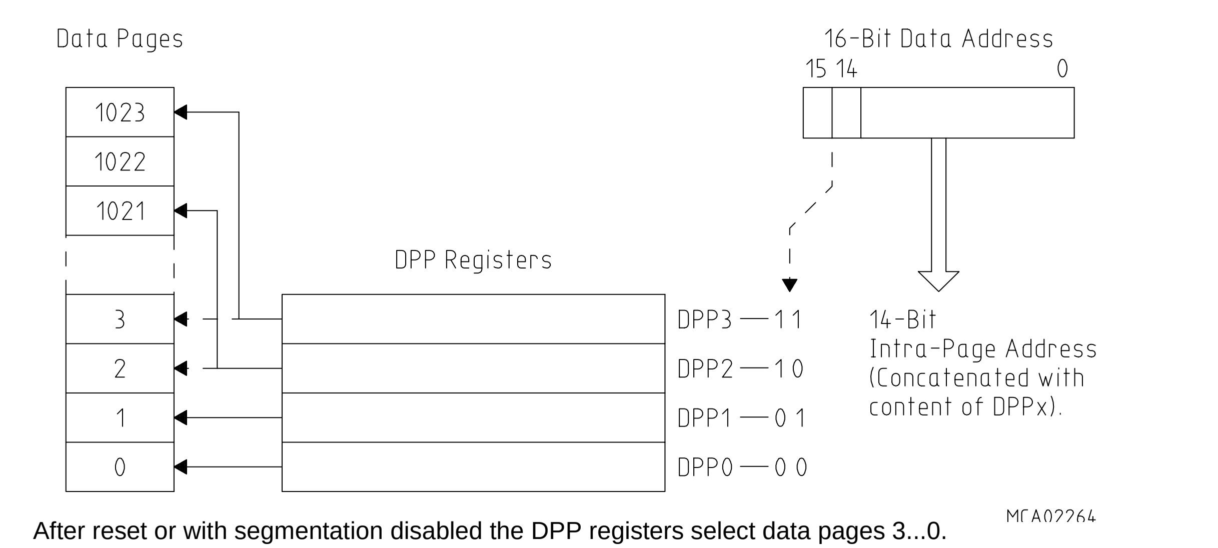
Data paging is performed by concatenating the lower 14 bits of an indirect or direct long 16-bit address with the contents of the DPP register selected by the upper two bits of the 16-bit address. The contents of the selected DPP register specify one of the 1024 possible data pages. This data page base address together with the 14-bit page offset forms the physical 24-bit address (selectable part is driven to the address pins).

In case of non-segmented memory mode, only the two least significant bits of the implicitly selected DPP register are used to generate the physical address. Thus, extreme care should be taken when changing the content of a DPP register, if a non-segmented memory model is selected, because otherwise unexpected results could occur.

In case of the segmented memory mode the selected number of segment address bits (via bitfield SALSEL) of the respective DPP register is output on the respective segment address pins of Port 4 for all external data accesses.

A DPP register can be updated via any instruction, which is capable of modifying an SFR.

Note: Due to the internal instruction pipeline, a new DPP value is not yet usable for the operand address calculation of the instruction immediately following the instruction updating the DPP register.



After reset or with segmentation disabled the DPP registers select data pages 3...0.
All of the internal memory is accessible in these cases.

Figure 4-6 Addressing via the Data Page Pointers

The Context Pointer CP

This non-bit addressable register is used to select the current register context. This means that the CP register value determines the address of the first General Purpose Register (GPR) within the current register bank of up to 16 wordwide and/or bytewide GPRs.

CP
Context Pointer **SFR (FE10$_H$/08$_H$)** **Reset value: FC00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	**1**	**1**	**1**	**cp**											**0**
r	r	r	r	rw											r

Bit	Function
cp	**Modifiable portion of register CP** Specifies the (word) base address of the current register bank. When writing a value to register CP with bits CP.11...CP.9 = ‘000’, bits CP.11...CP.10 are set to ‘11’ by hardware, in all other cases all bits of bit field “cp” receive the written value.

Note: It is the user’s responsibility that the physical GPR address specified via CP register plus short GPR address must always be an internal RAM location. If this condition is not met, unexpected results may occur.

- Do not set CP below the IRAM start address, i.e. 00’FA00$_H$/00’F600$_H$/00’F200$_H$ (referring to an IRAM size of 1/2/3 KByte)
- Do not set CP above 00’FDFE$_H$
- Be careful using the upper GPRs with CP above 00’FDE0$_H$

The CP register can be updated via any instruction which is capable of modifying an SFR.

Note: Due to the internal instruction pipeline, a new CP value is not yet usable for GPR address calculations of the instruction immediately following the instruction updating the CP register.

The Switch Context instruction (SCXT) allows to save the content of register CP on the stack and updating it with a new value in just one machine cycle.

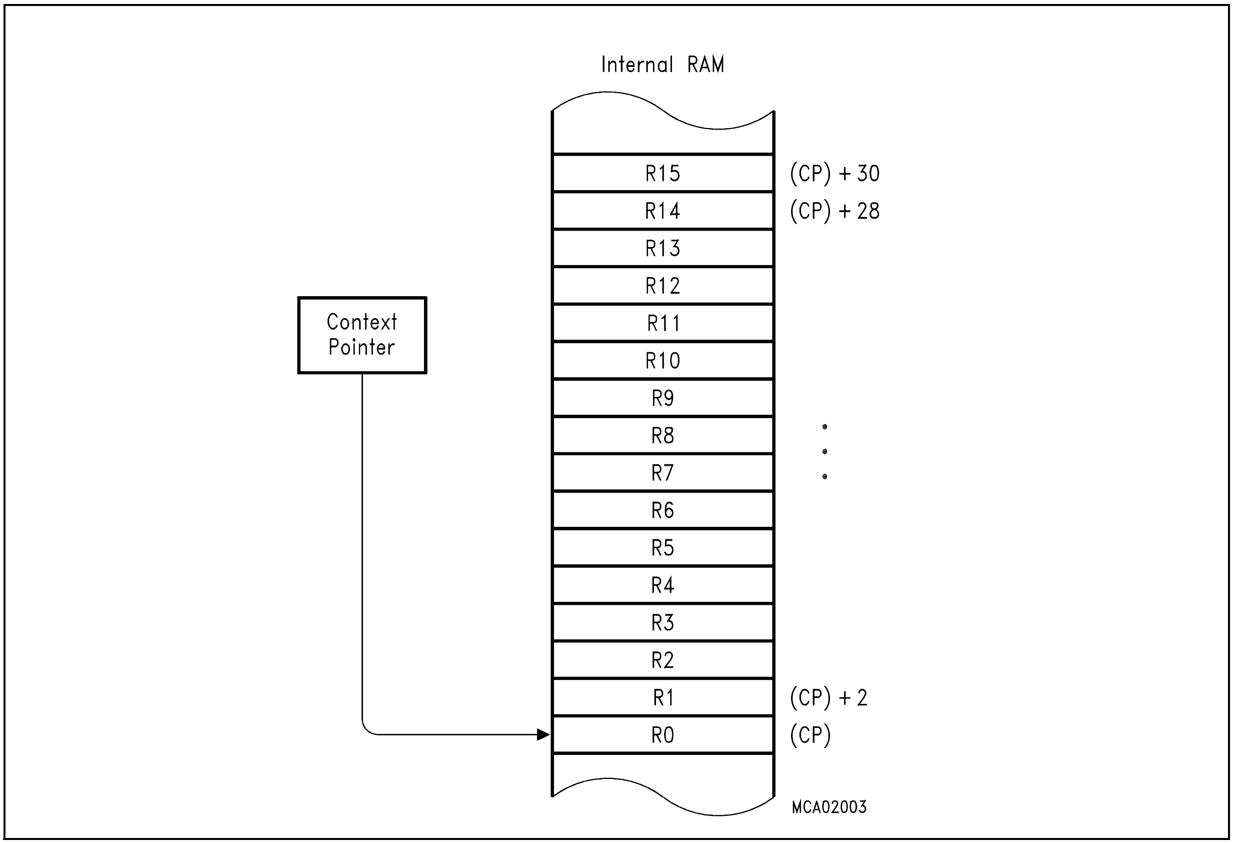


Figure 4-7 Register Bank Selection via Register CP

Several addressing modes use register CP implicitly for address calculations. The addressing modes mentioned below are described in chapter “Instruction Set Summary”.

Short 4-Bit GPR Addresses (mnemonic: Rw or Rb) specify an address relative to the memory location specified by the contents of the CP register, i.e. the base of the current register bank.
Depending on whether a relative word (Rw) or byte (Rb) GPR address is specified, the short 4-bit GPR address is either multiplied by two or not before it is added to the content of register CP (see figure below). Thus, both byte and word GPR accesses are possible in this way.

GPRs used as indirect address pointers are always accessed wordwise. For some instructions only the first four GPRs can be used as indirect address pointers. These GPRs are specified via short 2-bit GPR addresses. The respective physical address calculation is identical to that for the short 4-bit GPR addresses.

Short 8-Bit Register Addresses (mnemonic: reg or bitoff) within a range from $F0_H$ to FF_H interpret the four least significant bits as short 4-bit GPR address, while the four most significant bits are ignored. The respective physical GPR address calculation is identical to that for the short 4-bit GPR addresses. For single bit accesses on a GPR, the GPR's word address is calculated as just described, but the position of the bit within the word is specified by a separate additional 4-bit value.

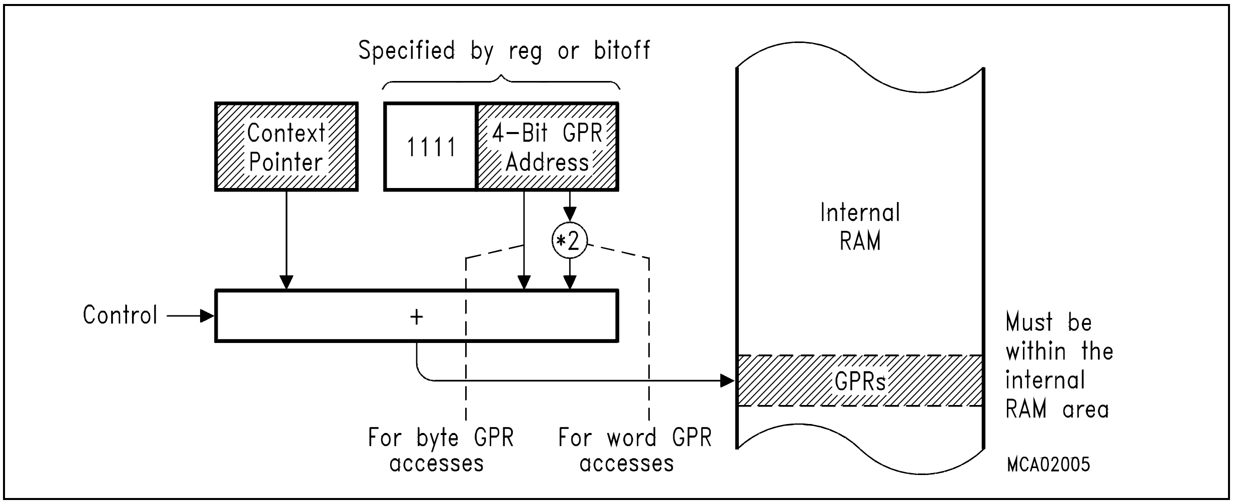


Figure 4-8 Implicit CP Use by Short GPR Addressing Modes

The Stack Pointer SP

This non-bit addressable register is used to point to the top of the internal system stack (TOS). The SP register is pre-decremented whenever data is to be pushed onto the stack, and it is post-incremented whenever data is to be popped from the stack. Thus, the system stack grows from higher toward lower memory locations.

Since the least significant bit of register SP is tied to ’0’ and bits 15 through 12 are tied to ’1’ by hardware, the SP register can only contain values from F000$_H$ to FFFE$_H$. This allows to access a physical stack within the internal RAM of the C164. A virtual stack (usually bigger) can be realized via software. This mechanism is supported by registers STKOV and STKUN (see respective descriptions below).

The SP register can be updated via any instruction, which is capable of modifying an SFR.

Note: Due to the internal instruction pipeline, a POP or RETURN instruction must not immediately follow an instruction updating the SP register.

SP

Stack Pointer Register **SFR (FE12$_H$/09$_H$)** **Reset value: FC00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	**1**	**1**	**1**	**sp**											**0**
r	r	r	r	rwh											r

Bit	Function
sp	**Modifiable portion of register SP** Specifies the top of the internal system stack.

The Stack Overflow Pointer STKOV

This non-bit addressable register is compared against the SP register after each operation, which pushes data onto the system stack (e.g. PUSH and CALL instructions or interrupts) and after each subtraction from the SP register. If the content of the SP register is less than the content of the STKOV register, a stack overflow hardware trap will occur.

Since the least significant bit of register STKOV is tied to ’0’ and bits 15 through 12 are tied to ’1’ by hardware, the STKOV register can only contain values from $F000_H$ to $FFFE_H$.

STKOV
Stack Overflow Reg. **SFR ($FE14_H$/$0A_H$)** **Reset value:$FA00_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	**1**	**1**	**1**	**stkov**											**0**
r	r	r	r	rw											

Bit	Function
stkov	**Modifiable portion of register STKOV** Specifies the lower limit of the internal system stack.

The Stack Overflow Trap (entered when (SP) < (STKOV)) may be used in two different ways:

- **Fatal error indication** treats the stack overflow as a system error through the associated trap service routine. Under these circumstances data in the bottom of the stack may have been overwritten by the status information stacked upon servicing the stack overflow trap.
- **Automatic system stack flushing** allows to use the system stack as a ’Stack Cache’ for a bigger external user stack. In this case register STKOV should be initialized to a value, which represents the desired lowest Top of Stack address plus 12 according to the selected maximum stack size. This considers the worst case that will occur, when a stack overflow condition is detected just during entry into an interrupt service routine. Then, six additional stack word locations are required to push IP, PSW, and CSP for both the interrupt service routine and the hardware trap service routine.

More details about the stack overflow trap service routine and virtual stack management are given in chapter “System Programming”.

The Stack Underflow Pointer STKUN

This non-bit addressable register is compared against the SP register after each operation, which pops data from the system stack (e.g. POP and RET instructions) and after each addition to the SP register. If the content of the SP register is greater than the the content of the STKUN register, a stack underflow hardware trap will occur.

Since the least significant bit of register STKUN is tied to ’0’ and bits 15 through 12 are tied to ’1’ by hardware, the STKUN register can only contain values from F000$_H$ to FFFE$_H$.

STKUN

Stack Underflow Reg. **SFR (FE16$_H$/0B$_H$)** **Reset value: FC00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	**1**	**1**	**1**	**stkun**											**0**
r	r	r	r	rw											r

Bit	Function
stkun	**Modifiable portion of register STKUN** Specifies the upper limit of the internal system stack.

The Stack Underflow Trap (entered when (SP) > (STKUN)) may be used in two different ways:

- **Fatal error indication** treats the stack underflow as a system error through the associated trap service routine.
- **Automatic system stack refilling** allows to use the system stack as a ’Stack Cache’ for a bigger external user stack. In this case register STKUN should be initialized to a value, which represents the desired highest Bottom of Stack address.

More details about the stack underflow trap service routine and virtual stack management are given in chapter “System Programming”.

Scope of Stack Limit Control

The stack limit control realized by the register pair STKOV and STKUN detects cases where the stack pointer SP is moved outside the defined stack area either by ADD or SUB instructions or by PUSH or POP operations (explicit or implicit, i.e. CALL or RET instructions).

This control mechanism is not triggered, i.e. no stack trap is generated, when

- the stack pointer SP is directly updated via MOV instructions
- the limits of the stack area (STKOV, STKUN) are changed, so that SP is outside of the new limits.

The Multiply/Divide High Register MDH

This register is a part of the 32-bit multiply/divide register, which is implicitly used by the CPU, when it performs a multiplication or a division. After a multiplication, this non-bit addressable register represents the high order 16 bits of the 32-bit result. For long divisions, the MDH register must be loaded with the high order 16 bits of the 32-bit dividend before the division is started. After any division, register MDH represents the 16-bit remainder.

MDH

Multiply/Divide High Reg. **SFR (FE0C$_H$/06$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
mdh															
rwh															

Bit	Function
mdh	Specifies the high order 16 bits of the 32-bit multiply and divide reg. MD.

Whenever this register is updated via software, the Multiply/Divide Register In Use (MDRIU) flag in the Multiply/Divide Control register (MDC) is set to ’1’.

When a multiplication or division is interrupted before its completion and when a new multiply or divide operation is to be performed within the interrupt service routine, register MDH must be saved along with registers MDL and MDC to avoid erroneous results.

A detailed description of how to use the MDH register for programming multiply and divide algorithms can be found in chapter “System Programming”.

The Multiply/Divide Low Register MDL

This register is a part of the 32-bit multiply/divide register, which is implicitly used by the CPU, when it performs a multiplication or a division. After a multiplication, this non-bit addressable register represents the low order 16 bits of the 32-bit result. For long divisions, the MDL register must be loaded with the low order 16 bits of the 32-bit dividend before the division is started. After any division, register MDL represents the 16-bit quotient.

MDL
Multiply/Divide Low Reg. **SFR (FE0E$_H$/07$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
mdl															
rwh															

Bit	Function
mdl	Specifies the low order 16 bits of the 32-bit multiply and divide reg. MD.

Whenever this register is updated via software, the Multiply/Divide Register In Use (MDRIU) flag in the Multiply/Divide Control register (MDC) is set to ’1’. The MDRIU flag is cleared, whenever the MDL register is read via software.

When a multiplication or division is interrupted before its completion and when a new multiply or divide operation is to be performed within the interrupt service routine, register MDL must be saved along with registers MDH and MDC to avoid erroneous results.

A detailed description of how to use the MDL register for programming multiply and divide algorithms can be found in chapter “System Programming”.

The Multiply/Divide Control Register MDC

This bit addressable 16-bit register is implicitly used by the CPU, when it performs a multiplication or a division. It is used to store the required control information for the corresponding multiply or divide operation. Register MDC is updated by hardware during each single cycle of a multiply or divide instruction.

MDC
Multiply/Divide Control Reg. **SFR ($FF0E_H$/87_H)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**!!**	**!!**	**!!**	**MDRIU**	**!!**	**!!**	**!!**	**!!**
-	-	-	-	-	-	-	-	r(w)h	r(w)h	r(w)h	r(w)h	r(w)h	r(w)h	r(w)h	r(w)h

Bit	Function
MDRIU	**Multiply/Divide Register In Use** 0: Cleared, when register MDL is read via software. 1: Set when register MDL or MDH is written via software, or when a multiply or divide instruction is executed.
!!	**Internal Machine Status** The multiply/divide unit uses these bits to control internal operations. Never modify these bits without saving and restoring register MDC.

When a division or multiplication was interrupted before its completion and the multiply/divide unit is required, the MDC register must first be saved along with registers MDH and MDL (to be able to restart the interrupted operation later), and then it must be cleared prepare it for the new calculation. After completion of the new division or multiplication, the state of the interrupted multiply or divide operation must be restored.

The MDRIU flag is the only portion of the MDC register which might be of interest for the user. The remaining portions of the MDC register are reserved for dedicated use by the hardware, and should never be modified by the user in another way than described above. Otherwise, a correct continuation of an interrupted multiply or divide operation cannot be guaranteed.

A detailed description of how to use the MDC register for programming multiply and divide algorithms can be found in chapter “System Programming”.

The Constant Zeros Register ZEROS

All bits of this bit-addressable register are fixed to ’0’ by hardware. This register can be read only. Register ZEROS can be used as a register-addressable constant of all zeros, i.e. for bit manipulation or mask generation. It can be accessed via any instruction, which is capable of addressing an SFR.

ZEROS
Zeros Register **SFR (FF1C$_H$/8E$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

The Constant Ones Register ONES

All bits of this bit-addressable register are fixed to ’1’ by hardware. This register can be read only. Register ONES can be used as a register-addressable constant of all ones, i.e. for bit manipulation or mask generation. It can be accessed via any instruction, which is capable of addressing an SFR.

ONES
Ones Register **SFR (FF1E$_H$/8F$_H$)** **Reset Value: FFFF$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

5 Interrupt and Trap Functions

The architecture of the C164 supports several mechanisms for fast and flexible response to service requests that can be generated from various sources internal or external to the microcontroller.

These mechanisms include:

Normal Interrupt Processing

The CPU temporarily suspends the current program execution and branches to an interrupt service routine in order to service an interrupt requesting device. The current program status (IP, PSW, in segmentation mode also CSP) is saved on the internal system stack. A prioritization scheme with 16 priority levels allows the user to specify the order in which multiple interrupt requests are to be handled.

Interrupt Processing via the Peripheral Event Controller (PEC)

A faster alternative to normal software controlled interrupt processing is servicing an interrupt requesting device with the C164's integrated Peripheral Event Controller (PEC). Triggered by an interrupt request, the PEC performs a single word or byte data transfer between any two locations in segment 0 (data pages 0 through 3) through one of eight programmable PEC Service Channels. During a PEC transfer the normal program execution of the CPU is halted for just 1 instruction cycle. No internal program status information needs to be saved. The same prioritization scheme is used for PEC service as for normal interrupt processing. PEC transfers share the 2 highest priority levels.

Trap Functions

Trap functions are activated in response to special conditions that occur during the execution of instructions. A trap can also be caused externally by the Non-Maskable Interrupt pin $\overline{\text{NMI}}$. Several hardware trap functions are provided for handling erroneous conditions and exceptions that arise during the execution of an instruction. Hardware traps always have highest priority and cause immediate system reaction. The software trap function is invoked by the TRAP instruction, which generates a software interrupt for a specified interrupt vector. For all types of traps the current program status is saved on the system stack.

External Interrupt Processing

Although the C164 does not provide dedicated interrupt pins, it allows to connect external interrupt sources and provides several mechanisms to react on external events, including standard inputs, non-maskable interrupts and fast external interrupts. These interrupt functions are alternate port functions, except for the non-maskable interrupt and the reset input.

5.1 Interrupt System Structure

The C164 provides 40 separate interrupt nodes that may be assigned to 16 priority levels. In order to support modular and consistent software design techniques, most sources of an interrupt or PEC request are supplied with a separate interrupt control register and interrupt vector. The control register contains the interrupt request flag, the interrupt enable bit, and the interrupt priority of the associated source. Each source request is then activated by one specific event, depending on the selected operating mode of the respective device. For efficient usage of the resources also multi-source interrupt nodes are incorporated. These nodes can be activated by several source requests, e.g. as different kinds of errors in the serial interfaces. However, specific status flags which identify the type of error are implemented in the serial channels' control registers. Additional sharing of interrupt nodes is supported via the interrupt subnode control register ISNC (see description below).

The C164 provides a vectored interrupt system. In this system specific vector locations in the memory space are reserved for the reset, trap, and interrupt service functions. Whenever a request occurs, the CPU branches to the location that is associated with the respective interrupt source. This allows direct identification of the source that caused the request. The only exceptions are the class B hardware traps, which all share the same interrupt vector. The status flags in the Trap Flag Register (TFR) can then be used to determine which exception caused the trap. For the special software TRAP instruction, the vector address is specified by the operand field of the instruction, which is a seven bit trap number.

The reserved vector locations build a jump table in the low end of the C164's address space (segment 0). The jump table is made up of the appropriate jump instructions that transfer control to the interrupt or trap service routines, which may be located anywhere within the address space. The entries of the jump table are located at the lowest addresses in code segment 0 of the address space. Each entry occupies 2 words, except for the reset vector and the hardware trap vectors, which occupy 4 or 8 words. The table below lists all sources that are capable of requesting interrupt or PEC service in the C164, the associated interrupt vectors, their locations and the associated trap numbers. It also lists the mnemonics of the affected Interrupt Request flags and their corresponding Interrupt Enable flags. The mnemonics are composed of a part that specifies the respective source, followed by a part that specifies their function (IR=Interrupt Request flag, IE=Interrupt Enable flag).

Note: Each entry of the interrupt vector table provides room for two word instructions or one doubleword instruction. The respective vector location results from multiplying the trap number by 4 (4 bytes per entry).
All interrupt nodes that are currently not used by their associated modules or are not connected to a module in the actual derivative may be used to generate software controlled interrupt requests by setting the respective IR flag.

Table 5-1 C164 Interrupt Nodes and Vectors

Source of Interrupt or PEC Service Request	Request Flag	Enable Flag	Interrupt Vector	Vector Location	Trap Number
Fast External Interrupt 0	CC8IR	CC8IE	CC8INT	$00'0060_H$	18_H / 24_D
Fast External Interrupt 1	CC9IR	CC9IE	CC9INT	$00'0064_H$	19_H / 25_D
Fast External Interrupt 2	CC10IR	CC10IE	CC10INT	$00'0068_H$	$1A_H$ / 26_D
Fast External Interrupt 3	CC11IR	CC11IE	CC11INT	$00'006C_H$	$1B_H$ / 27_D
CAPCOM Register 16	CC16IR	CC16IE	CC16INT	$00'00C0_H$	30_H / 48_D
CAPCOM Register 17	CC17IR	CC17IE	CC17INT	$00'00C4_H$	31_H / 49_D
CAPCOM Register 18	CC18IR	CC18IE	CC18INT	$00'00C8_H$	32_H / 50_D
CAPCOM Register 19	CC19IR	CC19IE	CC19INT	$00'00CC_H$	33_H / 51_D
CAPCOM Register 20	CC20IR	CC20IE	CC20INT	$00'00D0_H$	34_H / 52_D
CAPCOM Register 21	CC21IR	CC21IE	CC21INT	$00'00D4_H$	35_H / 53_D
CAPCOM Register 22	CC22IR	CC22IE	CC22INT	$00'00D8_H$	36_H / 54_D
CAPCOM Register 23	CC23IR	CC23IE	CC23INT	$00'00DC_H$	37_H / 55_D
CAPCOM Register 24	CC24IR	CC24IE	CC24INT	$00'00E0_H$	38_H / 56_D
CAPCOM Register 25	CC25IR	CC25IE	CC25INT	$00'00E4_H$	39_H / 57_D
CAPCOM Register 26	CC26IR	CC26IE	CC26INT	$00'00E8_H$	$3A_H$ / 58_D
CAPCOM Register 27	CC27IR	CC27IE	CC27INT	$00'00EC_H$	$3B_H$ / 59_D
CAPCOM Register 28	CC28IR	CC28IE	CC28INT	$00'00F0_H$	$3C_H$ / 60_D
CAPCOM Register 29	CC29IR	CC29IE	CC29INT	$00'0110_H$	44_H / 68_D
CAPCOM Register 30	CC30IR	CC30IE	CC30INT	$00'0114_H$	45_H / 69_D
CAPCOM Register 31	CC31IR	CC31IE	CC31INT	$00'0118_H$	46_H / 70_D
CAPCOM Timer 7	T7IR	T7IE	T7INT	$00'00F4_H$	$3D_H$ / 61_D
CAPCOM Timer 8	T8IR	T8IE	T8INT	$00'00F8_H$	$3E_H$ / 62_D
GPT1 Timer 2	T2IR	T2IE	T2INT	$00'0088_H$	22_H / 34_D
GPT1 Timer 3	T3IR	T3IE	T3INT	$00'008C_H$	23_H / 35_D
GPT1 Timer 4	T4IR	T4IE	T4INT	$00'0090_H$	24_H / 36_D
A/D Conversion Complete	ADCIR	ADCIE	ADCINT	$00'00A0_H$	28_H / 40_D
A/D Overrun Error	ADEIR	ADEIE	ADEINT	$00'00A4_H$	29_H / 41_D
ASC0 Transmit	S0TIR	S0TIE	S0TINT	$00'00A8_H$	$2A_H$ / 42_D
ASC0 Transmit Buffer	S0TBIR	S0TBIE	S0TBINT	$00'011C_H$	47_H / 71_D
ASC0 Receive	S0RIR	S0RIE	S0RINT	$00'00AC_H$	$2B_H$ / 43_D
ASC0 Error	S0EIR	S0EIE	S0EINT	$00'00B0_H$	$2C_H$ / 44_D
SSC Transmit	SCTIR	SCTIE	SCTINT	$00'00B4_H$	$2D_H$ / 45_D
SSC Receive	SCRIR	SCRIE	SCRINT	$00'00B8_H$	$2E_H$ / 46_D

Table 5-1 C164 Interrupt Nodes and Vectors (cont'd)

Source of Interrupt or PEC Service Request	Request Flag	Enable Flag	Interrupt Vector	Vector Location	Trap Number
SSC Error	SCEIR	SCEIE	SCEINT	$00'00BC_H$	$2F_H$ / 47_D
CAN	XP0IR	XP0IE	XP0INT	$00'0100_H$	40_H / 64_D
DataFlash/EEPROM Termination or Error	XP1IR	XP1IE	XP1INT	$00'0104_H$	41_H / 65_D
PLL/OWD, RTC (via ISNC)	XP3IR	XP3IE	XP3INT	$00'010C_H$	43_H / 67_D
CAPCOM 6 Interrupt	CC6IR	CC6IE	CC6INT	$00'00FC_H$	$3F_H$ / 63_D
CAPCOM6 Timer 12	T12IR	T12IE	T12INT	$00'0134_H$	$4D_H$ / 77_D
CAPCOM6 Timer 13	T13IR	T13IE	T13INT	$00'0138_H$	$4E_H$ / 78_D
CAPCOM6 Emergency	CC6IR	CC6IE	CC6EINT	$00'013C_H$	$4F_H$ / 79_D

The table below lists the vector locations for hardware traps and the corresponding status flags in register TFR. It also lists the priorities of trap service for cases, where more than one trap condition might be detected within the same instruction. After any reset (hardware reset, software reset instruction SRST, or reset by watchdog timer overflow) program execution starts at the reset vector at location 00’0000$_H$. Reset conditions have priority over every other system activity and therefore have the highest priority (trap priority III).

Software traps may be initiated to any vector location between 00’0000$_H$ and 00’01FC$_H$. A service routine entered via a software TRAP instruction is always executed on the current CPU priority level which is indicated in bit field ILVL in register PSW. This means that routines entered via the software TRAP instruction can be interrupted by all hardware traps or higher level interrupt requests.

Table 5-2 Hardware Trap Summary

Exception Condition	Trap Flag	Trap Vector	Vector Location	Trap Number	Trap Prio
Reset Functions:					
Hardware Reset		RESET	00’0000$_H$	00$_H$	III
Software Reset		RESET	00’0000$_H$	00$_H$	III
Watchdog Timer Overflow		RESET	00’0000$_H$	00$_H$	III
Class A Hardware Traps:					
Non-Maskable Interrupt	NMI	NMITRAP	00’0008$_H$	02$_H$	II
Stack Overflow	STKOF	STOTRAP	00’0010$_H$	04$_H$	II
Stack Underflow	STKUF	STUTRAP	00’0018$_H$	06$_H$	II
Class B Hardware Traps:					
Undefined Opcode	UNDOPC	BTRAP	00’0028$_H$	0A$_H$	I
Protected Instruction Fault	PRTFLT	BTRAP	00’0028$_H$	0A$_H$	I
Illegal Word Operand Access	ILLOPA	BTRAP	00’0028$_H$	0A$_H$	I
Illegal Instruction Access	ILLINA	BTRAP	00’0028$_H$	0A$_H$	I
Illegal External Bus Access	ILLBUS	BTRAP	00’0028$_H$	0A$_H$	I
Reserved			[2C$_H$ – 3C$_H$]	[0B$_H$ – 0F$_H$]	
Software Traps: TRAP Instruction			Any [00’0000$_H$ – 00’01FC$_H$] in steps of 4$_H$	Any [00$_H$ – 7F$_H$]	Current CPU Priority

Normal Interrupt Processing and PEC Service

During each instruction cycle one out of all sources which require PEC or interrupt processing is selected according to its interrupt priority. This priority of interrupts and PEC requests is programmable in two levels. Each requesting source can be assigned to a specific priority. A second level (called “group priority”) allows to specify an internal order for simultaneous requests from a group of different sources on the same priority level. At the end of each instruction cycle the one source request with the highest current priority will be determined by the interrupt system. This request will then be serviced, if its priority is higher than the current CPU priority in register PSW.

Interrupt System Register Description

Interrupt processing is controlled globally by register PSW through a general interrupt enable bit (IEN) and the CPU priority field (ILVL). Additionally the different interrupt sources are controlled individually by their specific interrupt control registers (...IC). Thus, the acceptance of requests by the CPU is determined by both the individual interrupt control registers and the PSW. PEC services are controlled by the respective PECCx register and the source and destination pointers, which specify the task of the respective PEC service channel.

5.1.1 Interrupt Control Registers

All interrupt control registers are organized identically. The lower 8 bits of an interrupt control register contain the complete interrupt status information of the associated source, which is required during one round of prioritization, the upper 8 bits of the respective register are reserved.. All interrupt control registers are bit-addressable and all bits can be read or written via software. This allows each interrupt source to be programmed or modified with just one instruction. When accessing interrupt control registers through instructions which operate on word data types, their upper 8 bits (15...8) will return zeros, when read, and will discard written data.

The layout of the Interrupt Control registers shown below applies to each xxIC register, where xx stands for the mnemonic for the respective source.

xxIC
Interrupt Control Register **(E)SFR (yyyy$_H$/zz$_H$)** **Reset value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								xxIR	**xxIE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Bit	Function
GLVL	**Group Level** Defines the internal order for simultaneous requests of the same priority. 3: Highest group priority 0: Lowest group priority
ILVL	**Interrupt Priority Level** Defines the priority level for the arbitration of requests. F$_H$: Highest priority level 0$_H$: Lowest priority level
xxIE	**Interrupt Enable Control Bit** (individually enables/disables a specific source) 0: Interrupt request is disabled 1: Interrupt Request is enabled
xxIR	**Interrupt Request Flag** 0: No request pending 1: This source has raised an interrupt request

The **Interrupt Request Flag** is set by hardware whenever a service request from the respective source occurs. It is cleared automatically upon entry into the interrupt service routine or upon a PEC service. In the case of PEC service the Interrupt Request flag remains set, if the COUNT field in register PECCx of the selected PEC channel decrements to zero. This allows a normal CPU interrupt to respond to a completed PEC block transfer.

Note: Modifying the Interrupt Request flag via software causes the same effects as if it had been set or cleared by hardware.

Interrupt Priority Level and Group Level

The four bits of bit field ILVL specify the priority level of a service request for the arbitration of simultaneous requests. The priority increases with the numerical value of ILVL, so 0000_B is the lowest and 1111_B is the highest priority level.

When more than one interrupt request on a specific level gets active at the same time, the values in the respective bit fields GLVL are used for second level arbitration to select one request for being serviced. Again the group priority increases with the numerical value of GLVL, so 00_B is the lowest and 11_B is the highest group priority.

Note: All interrupt request sources that are enabled and programmed to the same priority level must always be programmed to different group priorities. Otherwise an incorrect interrupt vector will be generated.

Upon entry into the interrupt service routine, the priority level of the source that won the arbitration and who's priority level is higher than the current CPU level, is copied into bit field ILVL of register PSW after pushing the old PSW contents on the stack.

The interrupt system of the C164 allows nesting of up to 15 interrupt service routines of different priority levels (level 0 cannot be arbitrated).

Interrupt requests that are programmed to priority levels 15 or 14 (ie, ILVL=$111X_B$) will be serviced by the PEC, unless the COUNT field of the associated PECC register contains zero. In this case the request will instead be serviced by normal interrupt processing. Interrupt requests that are programmed to priority levels 13 through 1 will always be serviced by normal interrupt processing.

Note: Priority level 0000_B is the default level of the CPU. Therefore a request on level 0 will never be serviced, because it can never interrupt the CPU. However, an enabled interrupt request on level 0000_B will terminate the C164's Idle mode and reactivate the CPU.

For interrupt requests which are to be serviced by the PEC, the associated PEC channel number is derived from the respective ILVL (LSB) and GLVL (see figure below). So programming a source to priority level 15 (ILVL=1111_B) selects the PEC channel group 7...4, programming a source to priority level 14 (ILVL=1110_B) selects the PEC channel group 3...0. The actual PEC channel number is then determined by the group priority field GLVL.

Simultaneous requests for PEC channels are prioritized according to the PEC channel number, where channel 0 has lowest and channel 8 has highest priority.

Note: All sources that request PEC service must be programmed to different PEC channels. Otherwise an incorrect PEC channel may be activated.

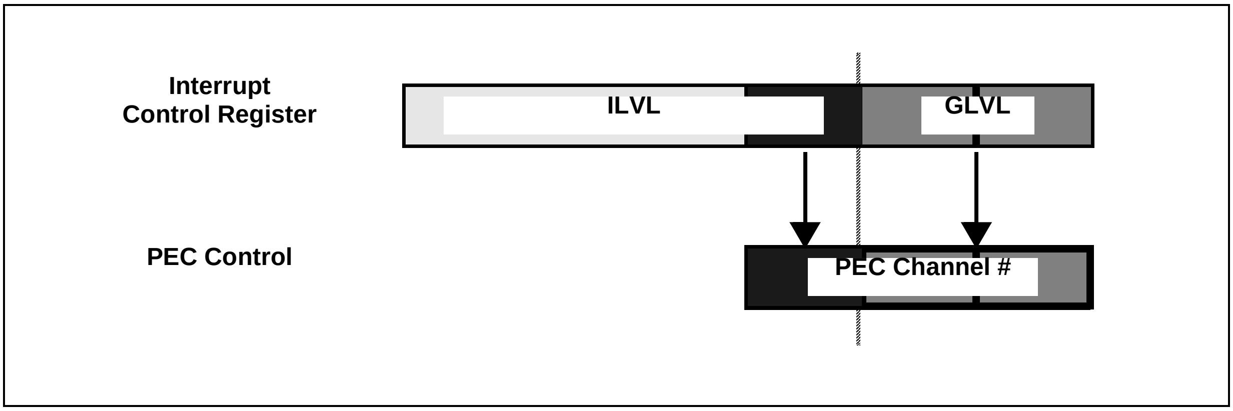


Figure 5-1 Priority Levels and PEC Channels

The table below shows in a few examples, which action is executed with a given programming of an interrupt control register.

Table 5-3 Interrupt Priority Examples

Priority Level		Type of Service	
ILVL	**GLVL**	**COUNT = 00H**	**COUNT ≠ 00_H**
1 1 1 **1**	**1 1**	CPU interrupt, level 15, group priority 3	PEC service, channel 7
1 1 1 **1**	**1 0**	CPU interrupt, level 15, group priority 2	PEC service, channel 6
1 1 1 **0**	**1 0**	CPU interrupt, level 14, group priority 2	PEC service, channel 2
1 1 0 1	1 0	CPU interrupt, level 13, group priority 2	CPU interrupt, level 13, group priority 2
0 0 0 1	1 1	CPU interrupt, level 1, group priority 3	CPU interrupt, level 1, group priority 3
0 0 0 1	0 0	CPU interrupt, level 1, group priority 0	CPU interrupt, level 1, group priority 0
0 0 0 0	X X	No service!	No service!

Note: All requests on levels 13...1 cannot initiate PEC transfers. They are always serviced by an interrupt service routine. No PECC register is associated and no COUNT field is checked.

Interrupt Control Functions in the PSW

The Processor Status Word (PSW) is functionally divided into 2 parts: the lower byte of the PSW basically represents the arithmetic status of the CPU, the upper byte of the PSW controls the interrupt system of the C164 and the arbitration mechanism for the external bus interface.

Note: Pipeline effects have to be considered when enabling/disabling interrupt requests via modifications of register PSW (see chapter "The Central Processing Unit").

PSW

Processor Status Word **SFR ($FF10_H/88_H$)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILVL				**IEN**	**-**	**-**	**-**	**-**	**USR0**	**MULIP**	**E**	**Z**	**V**	**C**	**N**
rwh				rw	-	-	-	-	rw	rwh	rwh	rwh	rwh	rwh	rwh

Bit	Function
N, C, V, Z, E, MULIP, USR0	**CPU status flags** (Described in section "The Central Processing Unit") Define the current status of the CPU (ALU, multiplication unit).
IEN	**Interrupt Enable Control Bit** (globally enables/disables interrupt requests) 0: Interrupt requests are disabled 1: Interrupt requests are enabled
ILVL	**CPU Priority Level** Defines the current priority level for the CPU F_H: Highest priority level 0_H: Lowest priority level

CPU Priority ILVL defines the current level for the operation of the CPU. This bit field reflects the priority level of the routine that is currently executed. Upon the entry into an interrupt service routine this bit field is updated with the priority level of the request that is being serviced. The PSW is saved on the system stack before. The CPU level determines the minimum interrupt priority level that will be serviced. Any request on the same or a lower level will not be acknowledged.
The current CPU priority level may be adjusted via software to control which interrupt request sources will be acknowledged.

PEC transfers do not really interrupt the CPU, but rather “steal” a single cycle, so PEC services do not influence the ILVL field in the PSW.

Hardware traps switch the CPU level to maximum priority (i.e. 15) so no interrupt or PEC requests will be acknowledged while an exception trap service routine is executed.

Note: The TRAP instruction does not change the CPU level, so software invoked trap service routines may be interrupted by higher requests.

Interrupt Enable bit IEN globally enables or disables PEC operation and the acceptance of interrupts by the CPU. When IEN is cleared, no new interrupt requests are accepted by the CPU. Requests that already have entered the pipeline at that time will process, however. When IEN is set to '1', all interrupt sources, which have been individually enabled by the interrupt enable bits in their associated control registers, are globally enabled.

Note: Traps are non-maskable and are therefore not affected by the IEN bit.

5.2 Operation of the PEC Channels

The C164's Peripheral Event Controller (PEC) provides 8 PEC service channels, which move a single byte or word between two locations in segment 0 (data pages 3...0). This is the fastest possible interrupt response and in many cases is sufficient to service the respective peripheral request (e.g. serial channels, etc.). Each channel is controlled by a dedicated PEC Channel Counter/Control register (PECCx) and a pair of pointers for source (SRCPx) and destination (DSTPx) of the data transfer.

The PECC registers control the action that is performed by the respective PEC channel.

PECCx
PEC Ch.x Ctrl. Reg. **SFR (FECy$_H$/6z$_H$, see table)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**-**	**-**	**-**	**-**	**INC**		**BWT**	**COUNT**							
-	-	-	-	-	rw		rw	rw							

Bit	Function
COUNT	**PEC Transfer Count** Counts PEC transfers and influences the channel's action (see table below)
BWT	**Byte / Word Transfer Selection** 0: Transfer a Word 1: Transfer a Byte
INC	**Increment Control** (Modification of SRCPx or DSTPx) 0 0: Pointers are not modified 0 1: Increment DSTPx by 1 or 2 (BWT) 1 0: Increment SRCPx by 1 or 2 (BWT) 1 1: Reserved. Do not use this combination. (changed to '10' by hardware)

Table 5-4 PEC Control Register Addresses

Register	Address	Reg. Space	Register	Address	Reg. Space
PECC0	FEC0$_H$ / 60$_H$	SFR	PECC4	FEC8$_H$ / 64$_H$	SFR
PECC1	FEC2$_H$ / 61$_H$	SFR	PECC5	FECA$_H$ / 65$_H$	SFR
PECC2	FEC4$_H$ / 62$_H$	SFR	PECC6	FECC$_H$ / 66$_H$	SFR
PECC3	FEC6$_H$ / 63$_H$	SFR	PECC7	FECE$_H$ / 67$_H$	SFR

Byte/Word Transfer bit BWT controls, if a byte or a word is moved during a PEC service cycle. This selection controls the transferred data size and the increment step for the modified pointer.

Increment Control Field INC controls, if one of the PEC pointers is incremented after the PEC transfer. It is not possible to increment both pointers, however. If the pointers are not modified (INC='00'), the respective channel will always move data from the same source to the same destination.

Note: The reserved combination '11' is changed to '10' by hardware. However, it is not recommended to use this combination.

The PEC Transfer Count Field COUNT controls the action of a respective PEC channel, where the content of bit field COUNT at the time the request is activated selects the action. COUNT may allow a specified number of PEC transfers, unlimited transfers or no PEC service at all.

The table below summarizes, how the COUNT field itself, the interrupt requests flag IR and the PEC channel action depends on the previous content of COUNT.

Table 5-5 Influence of Bitfield COUNT

Previous COUNT	Modified COUNT	IR after PEC service	Action of PEC Channel and Comments
FF_H	FF_H	'0'	Move a Byte / Word Continuous transfer mode, i.e. COUNT is not modified
$FE_H..02_H$	$FD_H..01_H$	'0'	Move a Byte / Word and decrement COUNT
01_H	00_H	'1'	Move a Byte / Word Leave request flag set, which triggers another request
00_H	00_H	('1')	**No action!** Activate interrupt service routine rather than PEC channel.

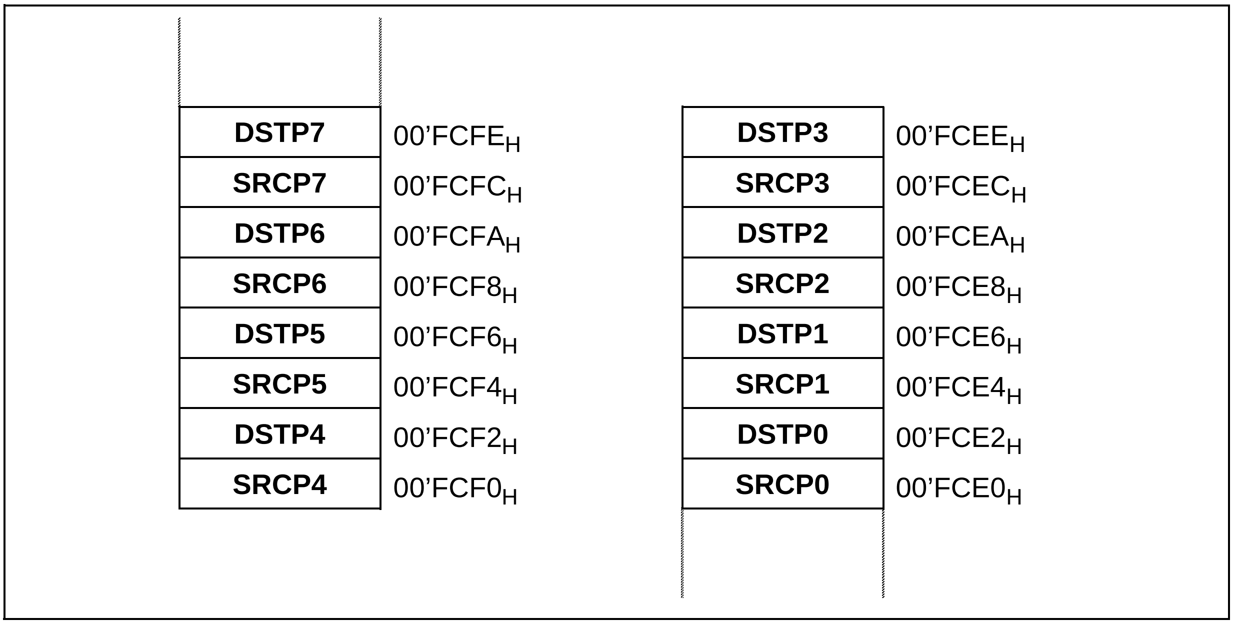
The PEC transfer counter allows to service a specified number of requests by the respective PEC channel, and then (when COUNT reaches 00_H) activate the interrupt service routine, which is associated with the priority level. After each PEC transfer the COUNT field is decremented and the request flag is cleared to indicate that the request has been serviced.

Continuous transfers are selected by the value FF$_H$ in bit field COUNT. In this case COUNT is not modified and the respective PEC channel services any request until it is disabled again.

When COUNT is decremented from 01$_H$ to 00$_H$ after a transfer, the request flag is not cleared, which generates another request from the same source. When COUNT already contains the value 00$_H$, the respective PEC channel remains idle and the associated interrupt service routine is activated instead. This allows to choose, if a level 15 or 14 request is to be serviced by the PEC or by the interrupt service routine.

Note: PEC transfers are only executed, if their priority level is higher than the CPU level, i.e. only PEC channels 7...4 are processed, while the CPU executes on level 14. All interrupt request sources that are enabled and programmed for PEC service should use different channels. Otherwise only one transfer will be performed for all simultaneous requests. When COUNT is decremented to 00$_H$, and the CPU is to be interrupted, an incorrect interrupt vector will be generated.

The source and destination pointers specifiy the locations between which the data is to be moved. A pair of pointers (SRCPx and DSTPx) is associated with each of the 8 PEC channels. These pointers do not reside in specific SFRs, but are mapped into the internal RAM of the C164 just below the bit-addressable area (see figure below).



DSTP7	00’FCFE$_H$	**DSTP3**	00’FCEE$_H$
SRCP7	00’FCFC$_H$	**SRCP3**	00’FCEC$_H$
DSTP6	00’FCFA$_H$	**DSTP2**	00’FCEA$_H$
SRCP6	00’FCF8$_H$	**SRCP2**	00’FCE8$_H$
DSTP5	00’FCF6$_H$	**DSTP1**	00’FCE6$_H$
SRCP5	00’FCF4$_H$	**SRCP1**	00’FCE4$_H$
DSTP4	00’FCF2$_H$	**DSTP0**	00’FCE2$_H$
SRCP4	00’FCF0$_H$	**SRCP0**	00’FCE0$_H$

Figure 5-2 Mapping of PEC Pointers into the Internal RAM

PEC data transfers do not use the data page pointers DPP3...DPP0. The PEC source and destination pointers are used as 16-bit intra-segment addresses within segment 0, so data can be transferred between any two locations within the first four data pages 3...0.

The pointer locations for inactive PEC channels may be used for general data storage. Only the required pointers occupy RAM locations.

Note: If word data transfer is selected for a specific PEC channel (i.e. BWT='0'), the respective source and destination pointers must both contain a valid word address which points to an even byte boundary. Otherwise the Illegal Word Access trap will be invoked, when this channel is used.

5.3 Prioritization of Interrupt and PEC Service Requests

Interrupt and PEC service requests from all sources can be enabled, so they are arbitrated and serviced (if they win), or they may be disabled, so their requests are disregarded and not serviced.

Enabling and disabling interrupt requests may be done via three mechanisms:

Control Bits allow to switch each individual source “ON” or “OFF”, so it may generate a request or not. The control bits (xxIE) are located in the respective interrupt control registers. All interrupt requests may be enabled or disabled generally via bit IEN in register PSW. This control bit is the “main switch” that selects, if requests from any source are accepted or not.
For a specific request to be arbitrated the respective source’s enable bit and the global enable bit must both be set.

The Priority Level automatically selects a certain group of interrupt requests that will be acknowledged, disclosing all other requests. The priority level of the source that won the arbitration is compared against the CPU’s current level and the source is only serviced, if its level is higher than the current CPU level. Changing the CPU level to a specific value via software blocks all requests on the same or a lower level. An interrupt source that is assigned to level 0 will be disabled and never be serviced.

The ATOMIC and EXTend instructions automatically disable all interrupt requests for the duration of the following 1...4 instructions. This is useful e.g. for semaphore handling and does not require to re-enable the interrupt system after the unseparable instruction sequence (see chapter “System Programming”).

Interrupt Class Management

An interrupt class covers a set of interrupt sources with the same importance, i.e. the same priority from the system’s viewpoint. Interrupts of the same class must not interrupt each other. The C164 supports this function with two features:

Classes with up to 4 members can be established by using the same interrupt priority (ILVL) and assigning a dedicated group level (GLVL) to each member. This functionality is built-in and handled automatically by the interrupt controller.

Classes with more than 4 members can be established by using a number of adjacent interrupt priorities (ILVL) and the respective group levels (4 per ILVL). Each interrupt service routine within this class sets the CPU level to the highest interrupt priority within the class. All requests from the same or any lower level are blocked now, i.e. no request of this class will be accepted.

The example below establishes 3 interrupt classes which cover 2 or 3 interrupt priorities, depending on the number of members in a class. A level 6 interrupt disables all other sources in class 2 by changing the current CPU level to 8, which is the highest priority (ILVL) in class 2. Class 1 requests or PEC requests are still serviced in this case.

The 24 interrupt sources (excluding PEC requests) are so assigned to 3 classes of priority rather than to 7 different levels, as the hardware support would do.

Table 5-6 Software controlled Interrupt Classes (Example)

ILVL	GLVL				Interpretation
(Priority)	3	2	1	0	
15					PEC service on up to 8 channels
14					
13					
12	X	X	X	X	Interrupt Class 1 5 sources on 2 levels
11	X				
10					
9					
8	X	X	X	X	Interrupt Class 2 9 sources on 3 levels
7	X	X	X	X	
6	X				
5	X	X	X	X	Interrupt Class 3 5 sources on 2 levels
4	X				
3					
2					
1					
0					No service!

5.4 Saving the Status during Interrupt Service

Before an interrupt request that has been arbitrated is actually serviced, the status of the current task is automatically saved on the system stack. The CPU status (PSW) is saved along with the location, where the execution of the interrupted task is to be resumed after returning from the service routine. This return location is specified through the Instruction Pointer (IP) and, in case of a segmented memory model, the Code Segment Pointer (CSP). Bit SGTDIS in register SYSCON controls, how the return location is stored.

The system stack receives the PSW first, followed by the IP (unsegmented) or followed by CSP and then IP (segmented mode). This optimizes the usage of the system stack, if segmentation is disabled.

The CPU priority field (ILVL in PSW) is updated with the priority of the interrupt request that is to be serviced, so the CPU now executes on the new level. If a multiplication or division was in progress at the time the interrupt request was acknowledged, bit MULIP in register PSW is set to ‘1’. In this case the return location that is saved on the stack is not the next instruction in the instruction flow, but rather the multiply or divide instruction itself, as this instruction has been interrupted and will be completed after returning from the service routine.

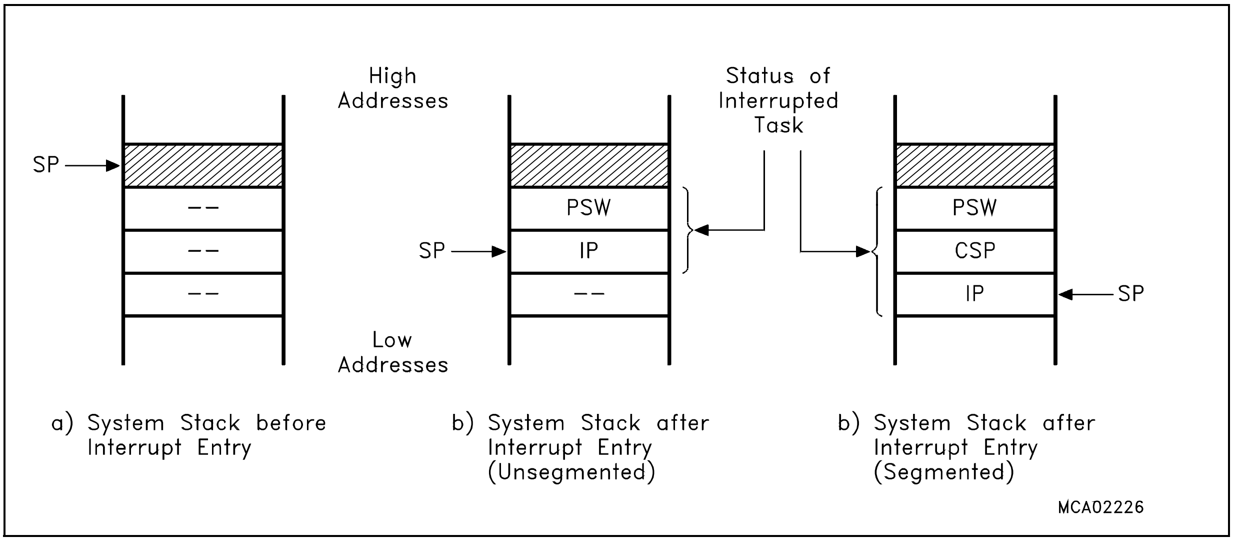


Figure 5-3 Task Status saved on the System Stack

The interrupt request flag of the source that is being serviced is cleared. The IP is loaded with the vector associated with the requesting source (the CSP is cleared in case of segmentation) and the first instruction of the service routine is fetched from the respective vector location, which is expected to branch to the service routine itself. The data page pointers and the context pointer are not affected.

When the interrupt service routine is left (RETI is executed), the status information is popped from the system stack in the reverse order, taking into account the value of bit SGTDIS.

Context Switching

An interrupt service routine usually saves all the registers it uses on the stack, and restores them before returning. The more registers a routine uses, the more time is wasted with saving and restoring. The C164 allows to switch the complete bank of CPU registers (GPRs) with a single instruction, so the service routine executes within its own, separate context.

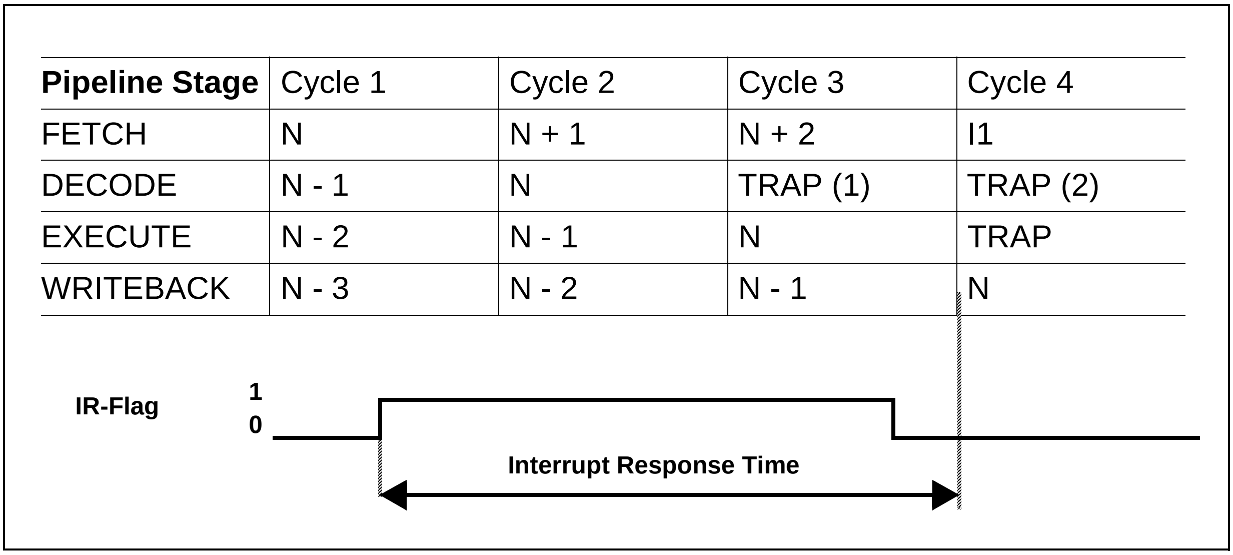
The instruction “SCXT CP, #New_Bank” pushes the content of the context pointer (CP) on the system stack and loads CP with the immediate value “New_Bank”, which selects a new register bank. The service routine may now use its “own registers”. This register bank is preserved, when the service routine terminates, i.e. its contents are available on the next call.
Before returning (RETI) the previous CP is simply POPped from the system stack, which returns the registers to the original bank.

Note: The first instruction following the SCXT instruction must not use a GPR.

Resources that are used by the interrupting program must eventually be saved and restored, e.g. the DPPs and the registers of the MUL/DIV unit.

5.5 Interrupt Response Times

The interrupt response time defines the time from an interrupt request flag of an enabled interrupt source being set until the first instruction (I1) being fetched from the interrupt vector location. The basic interrupt response time for the C164 is 3 instruction cycles.



Pipeline Stage	Cycle 1	Cycle 2	Cycle 3	Cycle 4
FETCH	N	N + 1	N + 2	I1
DECODE	N - 1	N	TRAP (1)	TRAP (2)
EXECUTE	N - 2	N - 1	N	TRAP
WRITEBACK	N - 3	N - 2	N - 1	N

Figure 5-4 Pipeline Diagram for Interrupt Response Time

All instructions in the pipeline including instruction N (during which the interrupt request flag is set) are completed before entering the service routine. The actual execution time for these instructions (e.g. waitstates) therefore influences the interrupt response time.

In the figure above the respective interrupt request flag is set in cycle 1 (fetching of instruction N). The indicated source wins the prioritization round (during cycle 2). In cycle 3 a TRAP instruction is injected into the decode stage of the pipeline, replacing instruction N+1 and clearing the source's interrupt request flag to '0'. Cycle 4 completes the injected TRAP instruction (save PSW, IP and CSP, if segmented mode) and fetches the first instruction (I1) from the respective vector location.

All instructions that entered the pipeline after setting of the interrupt request flag (N+1, N+2) will be executed after returning from the interrupt service routine.

The minimum interrupt response time is 5 states (10 TCL). This requires program execution from the internal code memory, no external operand read requests and setting the interrupt request flag during the last state of an instruction cycle. When the interrupt request flag is set during the first state of an instruction cycle, the minimum interrupt response time under these conditions is 6 state times (12 TCL).

The interrupt response time is increased by all delays of the instructions in the pipeline that are executed before entering the service routine (including N).

- When internal hold conditions between instruction pairs N-2/N-1 or N-1/N occur, or instruction N explicitly writes to the PSW or the SP, the minimum interrupt response time may be extended by 1 state time for each of these conditions.
- When instruction N reads an operand from the internal code memory, or when N is a call, return, trap, or MOV Rn, [Rm+ #data16] instruction, the minimum interrupt response time may additionally be extended by 2 state times during internal code memory program execution.
- In case instruction N reads the PSW and instruction N-1 has an effect on the condition flags, the interrupt response time may additionally be extended by 2 state times.

The worst case interrupt response time during internal code memory program execution adds to 12 state times (24 TCL).

Any reference to external locations increases the interrupt response time due to pipeline related access priorities. The following conditions have to be considered:

- Instruction fetch from an external location
- Operand read from an external location
- Result write-back to an external location

Depending on where the instructions, source and destination operands are located, there are a number of combinations. Note, however, that only access conflicts contribute to the delay.

A few examples illustrate these delays:

- The worst case interrupt response time including external accesses will occur, when instructions N, N+1 and N+2 are executed out of external memory, instructions N-1 and N require external operand read accesses, instructions N-3 through N write back external operands, and the interrupt vector also points to an external location. In this case the interrupt response time is the time to perform 9 word bus accesses, because instruction I1 cannot be fetched via the external bus until all write, fetch and read requests of preceding instructions in the pipeline are terminated.
- When the above example has the interrupt vector pointing into the internal code memory, the interrupt response time is 7 word bus accesses plus 2 states, because fetching of instruction I1 from internal code memory can start earlier.
- When instructions N, N+1 and N+2 are executed out of external memory and the interrupt vector also points to an external location, but all operands for instructions N-3 through N are in internal memory, then the interrupt response time is the time to perform 3 word bus accesses.
- When the above example has the interrupt vector pointing into the internal code memory, the interrupt response time is 1 word bus access plus 4 states.

After an interrupt service routine has been terminated by executing the RETI instruction, and if further interrupts are pending, the next interrupt service routine will not be entered until at least two instruction cycles have been executed of the program that was interrupted. In most cases two instructions will be executed during this time. Only one instruction will typically be executed, if the first instruction following the RETI instruction is a branch instruction (without cache hit), or if it reads an operand from internal code memory, or if it is executed out of the internal RAM.

Note: A bus access in this context includes all delays which can occur during an external bus cycle.

5.6 PEC Response Times

The PEC response time defines the time from an interrupt request flag of an enabled interrupt source being set until the PEC data transfer being started. The basic PEC response time for the C164 is 2 instruction cycles.

Pipeline Stage	Cycle 1	Cycle 2	Cycle 3	Cycle 4
FETCH	N	N + 1	N + 2	N + 2
DECODE	N - 1	N	PEC	N + 1
EXECUTE	N - 2	N - 1	N	PEC
WRITEBACK	N - 3	N - 2	N - 1	N

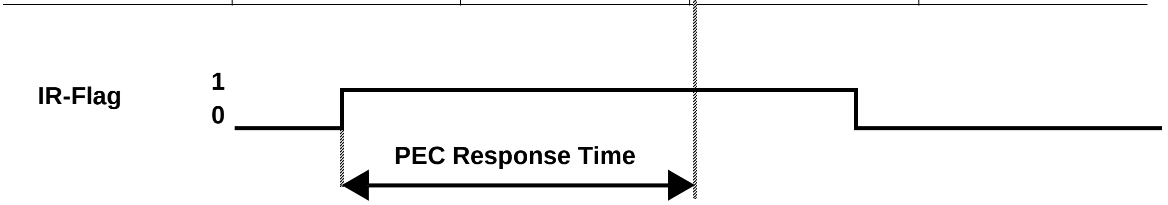


Figure 5-5 Pipeline Diagram for PEC Response Time

In the figure above the respective interrupt request flag is set in cycle 1 (fetching of instruction N). The indicated source wins the prioritization round (during cycle 2). In cycle 3 a PEC transfer "instruction" is injected into the decode stage of the pipeline, suspending instruction N+1 and clearing the source's interrupt request flag to '0'. Cycle 4 completes the injected PEC transfer and resumes the execution of instruction N+1.

All instructions that entered the pipeline after setting of the interrupt request flag (N+1, N+2) will be executed after the PEC data transfer.

Note: When instruction N reads any of the PEC control registers PECC7...PECC0, while a PEC request wins the current round of prioritization, this round is repeated and the PEC data transfer is started one cycle later.

The minimum PEC response time is 3 states (6 TCL). This requires program execution from the internal code memory, no external operand read requests and setting the interrupt request flag during the last state of an instruction cycle. When the interrupt request flag is set during the first state of an instruction cycle, the minimum PEC response time under these conditions is 4 state times (8 TCL).

The PEC response time is increased by all delays of the instructions in the pipeline that are executed before starting the data transfer (including N).

- When internal hold conditions between instruction pairs N-2/N-1 or N-1/N occur, the minimum PEC response time may be extended by 1 state time for each of these conditions.
- When instruction N reads an operand from the internal code memory, or when N is a call, return, trap, or MOV Rn, [Rm+ #data16] instruction, the minimum PEC response time may additionally be extended by 2 state times during internal code memory program execution.
- In case instruction N reads the PSW and instruction N-1 has an effect on the condition flags, the PEC response time may additionally be extended by 2 state times.

The worst case PEC response time during internal code memory program execution adds to 9 state times (18 TCL).

Any reference to external locations increases the PEC response time due to pipeline related access priorities. The following conditions have to be considered:

- Instruction fetch from an external location
- Operand read from an external location
- Result write-back to an external location

Depending on where the instructions, source and destination operands are located, there are a number of combinations. Note, however, that only access conflicts contribute to the delay.

A few examples illustrate these delays:

- The worst case interrupt response time including external accesses will occur, when instructions N and N+1 are executed out of external memory, instructions N-1 and N require external operand read accesses and instructions N-3, N-2 and N-1 write back external operands. In this case the PEC response time is the time to perform 7 word bus accesses.
- When instructions N and N+1 are executed out of external memory, but all operands for instructions N-3 through N-1 are in internal memory, then the PEC response time is the time to perform 1 word bus access plus 2 state times.

Once a request for PEC service has been acknowledged by the CPU, the execution of the next instruction is delayed by 2 state times plus the additional time it might take to fetch the source operand from internal code memory or external memory and to write the destination operand over the external bus in an external program environment.

Note: A bus access in this context includes all delays which can occur during an external bus cycle.

5.7 Interrupt Node Sharing

Interrupt nodes may be shared between several module requests either if the requests are generated mutually exclusive or if the requests are generated at a low rate. If more than one source is enabled in this case the interrupt handler will first have to determine the requesting source. However, this overhead is not critical for low rate requests.

This node sharing is controlled via the sub-node interrupt control register ISNC which provides a separate request flag and enable bit for each supported request source. The interrupt level used for arbitration is determined by the node control register (...IC).

The specific request flags within ISNC must be reset by software. If the respective request is likely to be activated at ro shortly after the time the request flag is cleared, the request flag should be cleared together with the corresponding enable bit. The enable bit can then be set again. This avoids undetected requests due to too short pulses at the interrupt node.

ISNC
Intr. Subnode Ctrl. Reg. **ESFR (F1DE$_H$/EF$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	**PLL IE**	**PLL IR**	**RTC IE**	**RTC IR**
-	-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw

Bit	Function
xxIR	**Interrupt Request Flag for Source xx** 0: No request from source xx pending. 1: Source xx has raised an interrupt request.
xxIE	**Interrupt Enable Control Bit for Source xx** 0: Source xx interrupt request is disabled. 1: Source xx interrupt request is enabled.

Table 5-7 Sub-node Control Bit Allocation

Bit pos.	Interrupt Source	Associated Node
15...4	Reserved.	Reserved.
3|2	PLL / OWD	XP3IC
1|0	RTC	XP3IC

Note: In order to ensure compatibility with other derivatives application software should never set reserved bits within register ISNC.

5.8 External Interrupts

Although the C164 has no dedicated INTR input pins, it provides many possibilities to react on external asynchronous events by using a number of IO lines for interrupt input. The interrupt function may either be combined with the pin's main function or may be used instead of it, i.e. if the main pin function is not required.

Interrupt signals may be connected to:

- EX3IN...EX0IN, the fast external interrupt input pins,
- CC27IO...CC24IO, capture input / compare output lines of the CAPCOM units,
- CC19IO...CC16IO, capture input / compare output lines of the CAPCOM units,
- T4IN, T2IN, the timer input pins

For each of these pins either a positive, a negative, or both a positive and a negative external transition can be selected to cause an interrupt or PEC service request. The edge selection is performed in the control register of the peripheral device associated with the respective port pin. The peripheral must be programmed to a specific operating mode to allow generation of an interrupt by the external signal. The priority of the interrupt request is determined by the interrupt control register of the respective peripheral interrupt source, and the interrupt vector of this source will be used to service the external interrupt request.

Note: In order to use any of the listed pins as external interrupt input, it must be switched to input mode via its direction control bit DPx.y in the respective port direction control register DPx.

Table 5-8 Pins to be used as External Interrupt Inputs

Port Pin	Original Function	Control Register
P1H.3-0/EX3-0IN	Fast external interrupt input pin	EXICON
P1H.7-4/CC27-24IO	CAPCOM Register 27-24 Capture Input	CC27-CC24
P8.3-0/CC19-16IO	CAPCOM Register 19-16 Capture Input	CC19-CC16
P5.6/T2IN	Auxiliary timer T2 input pin	T2CON
P5.7/T4IN	Auxiliary timer T4 input pin	T4CON

When port pins CCxIO are to be used as external interrupt input pins, bit field CCMODx in the control register of the corresponding capture/compare register CCx must select capture mode. When CCMODx is programmed to 001_B, the interrupt request flag CCxIR in register CCxIC will be set on a positive external transition at pin CCxIO. When CCMODx is programmed to 010_B, a negative external transition will set the interrupt request flag. When CCMODx=011_B, both a positive and a negative transition will set the request flag. In all three cases, the contents of the allocated CAPCOM timer will be latched into capture register CCx, independent whether the timer is running or not. When the interrupt enable bit CCxIE is set, a PEC request or an interrupt request for vector CCxINT will be generated.

Pins T2IN or T4IN can be used as external interrupt input pins when the associated auxiliary timer T2 or T4 in block GPT1 is configured for capture mode. This mode is selected by programming the mode control fields T2M or T4M in control registers T2CON or T4CON to 101_B. The active edge of the external input signal is determined by bit fields T2I or T4I. When these fields are programmed to $X01_B$, interrupt request flags T2IR or T4IR in registers T2IC or T4IC will be set on a positive external transition at pins T2IN or T4IN, respectively. When T2I or T4I are programmed to $X10_B$, then a negative external transition will set the corresponding request flag. When T2I or T4I are programmed to $X11_B$, both a positive and a negative transition will set the request flag. In all three cases, the contents of the core timer T3 will be captured into the auxiliary timer registers T2 or T4 based on the transition at pins T2IN or T4IN. When the interrupt enable bits T2IE or T4IE are set, a PEC request or an interrupt request for vector T2INT or T4INT will be generated.

Note: The non-maskable interrupt input pin $\overline{\text{NMI}}$ and the reset input $\overline{\text{RSTIN}}$ provide another possibility for the CPU to react on an external input signal. $\overline{\text{NMI}}$ and $\overline{\text{RSTIN}}$ are dedicated input pins, which cause hardware traps.

Fast External Interrupts

The input pins that may be used for external interrupts are sampled every 16 TCL, i.e. external events are scanned and detected in timeframes of 16 TCL. The C164 provides 8 interrupt inputs that are sampled every 2 TCL, so external events are captured faster than with standard interrupt inputs.

The lower 4 pins of Port P1H (P1H.3-P1H.0) can individually be programmed to this fast interrupt mode, where also the trigger transition (rising, falling or both) can be selected. The External Interrupt Control register EXICON controls this feature for all 4 pins.

EXICON
External Intr. Ctrl. Reg. **ESFR (F1C0$_H$/E0$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-		-		-		-		EXI3ES		EXI2ES		EXI1ES		EXI0ES	
-		-		-		-		rw		rw		rw		rw	

Bit	Function
EXIxES	**External Interrupt x Edge Selection Field (x=7...0)** 0 0: Fast external interrupts disabled: standard mode 0 1: Interrupt on positive edge (rising) 1 0: Interrupt on negative edge (falling) 1 1: Interrupt on any edge (rising or falling)

Note: The fast external interrupt inputs are sampled every 2 TCL. The interrupt request arbitration and processing, however, is executed every 8 TCL.

The interrupt control registers listed below (CC11IC...CC8IC) control the fast external interrupts of the C164. These fast external interrupt nodes and vectors are named according to the C167’s CAPCOM channels CC11...CC8, so interrupt nodes receive equal names throughout the architecture. See register description below.

CCxIC

CAPCOM x Intr. Ctrl. Reg. **SFR(See Table)** **Reset value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								CCxIR	**CCxIE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

Table 5-9 Fast External Interrupt Control Register Addresses

Register	**Address**	**External Interrupt**
CC8IC	$FF88_H$ / $C4_H$	EX0IN
CC9IC	$FF8A_H$ / $C5_H$	EX1IN
CC10IC	$FF8C_H$ / $C6_H$	EX2IN
CC11IC	$FF8E_H$ / $C7_H$	EX3IN

External Interrupt Source Control

The input source for the fast external interrupts (controlled via register EXICON) can be derived either from the associated port pin EXnIN or from an alternate source. This selection is controlled via register EXISEL.

Activating the alternate input source e.g. allows the detection of transitions on the interface lines of disabled interfaces. Upon this trigger the respective interface can be reactivated and respond to the detected activity.

EXISEL
Ext. Intr. Source Ctrl. Reg. **ESFR(F1DA$_H$/ED$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXI7SS		EXI6SS		EXI5SS		EXI4SS		EXI3SS		EXI2SS		EXI1SS		EXI0SS	
rw		rw		rw		rw		rw		rw		rw		rw	

Bit	Function
EXI0SS	**External Interrupt 0 Source Selection Field** 0 0: Input from associated EXzIN pin. 0 1: Input from alternate pin. 1 0: Input from EXzIN pin ORed with alternate pin. 1 1: Input from EXzIN pin ANDed with alternate pin.

The table below summarizes the association of the bitfields of register EXISEL with the respective interface input lines.

Table 5-10 Connection of Interface Inputs to External Interrupt Nodes

Bitfield	Associated Interface Line	Notes
EX0IN	CAN_RxD	CAN (C164CH, C164CG) The used pin depends on the assignment for the module.
EX2IN	RxD0	ASC0
EX3IN	SCLK	SSC

External Interrupts During Sleep Mode

During Sleep mode all peripheral clock signals are deactivated which also disables the standard edge detection logic for the fast external interrupts. However, transitions on these interrupt inputs must be recognized in order to initiate the wakeup. Therefore during Sleep mode a special edge detection logic for the fast external interrupts (EXzIN) is activated, which requires no clock signal (therefore also works in Sleep mode) and is equipped with an analog noise filter. This filter suppresses spikes (generated by noise) up to 10 ns. Input pulses with a duration of 100 ns minimum are recognized and generate an interrupt request.

This filter delays the recognition of an external wakeup signal by approx. 100 ns, but the spike suppression ensures safe and robust operation of the sleep/wakeup mechanism in an active environment.

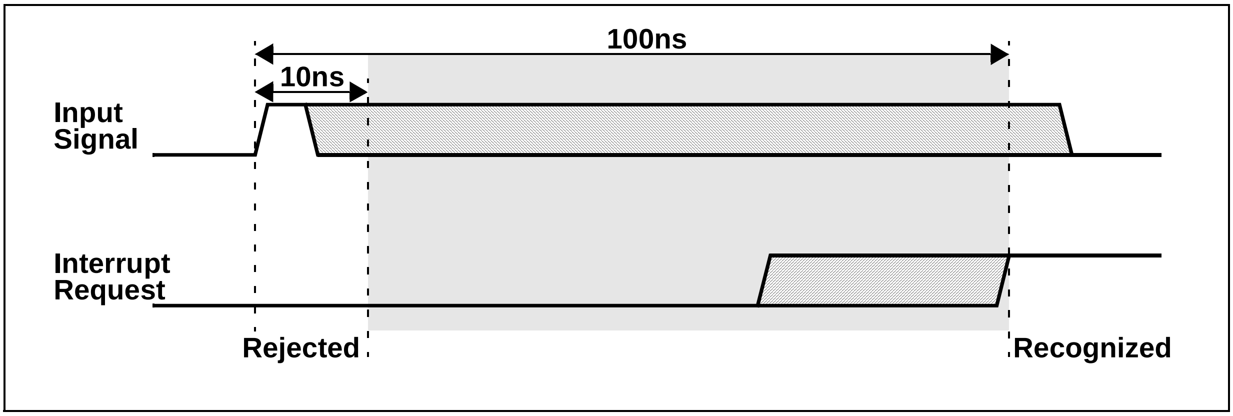


Figure 5-6 Input Noise Filter Operation

5.9 Trap Functions

Traps interrupt the current execution similar to standard interrupts. However, trap functions offer the possibility to bypass the interrupt system's prioritization process in cases where immediate system reaction is required. Trap functions are not maskable and always have priority over interrupt requests on any priority level.

The C164 provides two different kinds of trapping mechanisms. **Hardware traps** are triggered by events that occur during program execution (e.g. illegal access or undefined opcode), **software traps** are initiated via an instruction within the current execution flow.

Software Traps

The TRAP instruction is used to cause a software call to an interrupt service routine. The trap number that is specified in the operand field of the trap instruction determines which vector location in the address range from 00'0000$_H$ through 00'01FC$_H$ will be branched to.

Executing a TRAP instruction causes a similar effect as if an interrupt at the same vector had occurred. PSW, CSP (in segmentation mode), and IP are pushed on the internal system stack and a jump is taken to the specified vector location. When segmentation is enabled and a trap is executed, the CSP for the trap service routine is set to code segment 0. No Interrupt Request flags are affected by the TRAP instruction. The interrupt service routine called by a TRAP instruction must be terminated with a RETI (return from interrupt) instruction to ensure correct operation.

Note: The CPU level in register PSW is not modified by the TRAP instruction, so the service routine is executed on the same priority level from which it was invoked. Therefore, the service routine entered by the TRAP instruction can be interrupted by other traps or higher priority interrupts, other than when triggered by a hardware trap.

Hardware Traps

Hardware traps are issued by faults or specific system states that occur during runtime of a program (not identified at assembly time). A hardware trap may also be triggered intentionally, e.g. to emulate additional instructions by generating an Illegal Opcode trap. The C164 distinguishes eight different hardware trap functions. When a hardware trap condition has been detected, the CPU branches to the trap vector location for the respective trap condition. Depending on the trap condition, the instruction which caused the trap is either completed or cancelled (i.e. it has no effect on the system state) before the trap handling routine is entered.

Hardware traps are non-maskable and always have priority over every other CPU activity. If several hardware trap conditions are detected within the same instruction cycle, the highest priority trap is serviced (see table in section “Interrupt System Structure”).

PSW, CSP (in segmentation mode), and IP are pushed on the internal system stack and the CPU level in register PSW is set to the highest possible priority level (i.e. level 15), disabling all interrupts. The CSP is set to code segment zero, if segmentation is enabled. A trap service routine must be terminated with the RETI instruction.

The eight hardware trap functions of the C164 are divided into two classes:

Class A traps are
- external Non-Maskable Interrupt ($\overline{\text{NMI}}$)
- Stack Overflow
- Stack Underflow trap

These traps share the same trap priority, but have an individual vector address.

Class B traps are
- Undefined Opcode
- Protection Fault
- Illegal Word Operand Access
- Illegal Instruction Access
- Illegal External Bus Access Trap

These traps share the same trap priority, and the same vector address.

The bit-addressable Trap Flag Register (TFR) allows a trap service routine to identify the kind of trap which caused the exception. Each trap function is indicated by a separate request flag. When a hardware trap occurs, the corresponding request flag in register TFR is set to '1'.

The reset functions (hardware, software, watchdog) may be regarded as a type of trap. Reset functions have the highest system priority (trap priority III).

Class A traps have the second highest priority (trap priority II), on the 3rd rank are class B traps, so a class A trap can interrupt a class B trap. If more than one class A trap occur at a time, they are prioritized internally, with the NMI trap on the highest and the stack underflow trap on the lowest priority.

All class B traps have the same trap priority (trap priority I). When several class B traps get active at a time, the corresponding flags in the TFR register are set and the trap service routine is entered. Since all class B traps have the same vector, the priority of service of simultaneously occurring class B traps is determined by software in the trap service routine.

A class A trap occurring during the execution of a class B trap service routine will be serviced immediately. During the execution of a class A trap service routine, however, any class B trap occurring will not be serviced until the class A trap service routine is exited with a RETI instruction. In this case, the occurrence of the class B trap condition is stored in the TFR register, but the IP value of the instruction which caused this trap is lost.

TFR

Trap Flag Register **SFR (FFAC$_H$/D6$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NMI	STK OF	STK UF	-	-	-	-	-	UND OPC	-	-	-	PRT FLT	ILL OPA	ILL INA	ILL BUS
rwh	rwh	rwh		-	-	-	-	rwh	-	-	-	rwh	rwh	rwh	rwh

Bit	Function
ILLBUS	**Illegal External Bus Access Flag** An external access has been attempted with no external bus defined.
ILLINA	**Illegal Instruction Access Flag** A branch to an odd address has been attempted.
ILLOPA	**Illegal Word Operand Access Flag** A word operand access (read or write) to an odd address has been attempted.
PRTFLT	**Protection Fault Flag** A protected instruction with an illegal format has been detected.
UNDOPC	**Undefined Opcode Flag** The currently decoded instruction has no valid C164 opcode.
STKUF	**Stack Underflow Flag** The current stack pointer value exceeds the content of register STKUN.
STKOF	**Stack Overflow Flag** The current stack pointer value falls below the content of reg. STKOV.
NMI	**Non Maskable Interrupt Flag** A negative transition (falling edge) has been detected on pin $\overline{\text{NMI}}$.

Note: The trap service routine must clear the respective trap flag, otherwise a new trap will be requested after exiting the service routine. Setting a trap request flag by software causes the same effects as if it had been set by hardware.

In the case where e.g. an Undefined Opcode trap (class B) occurs simultaneously with an NMI trap (class A), both the NMI and the UNDOPC flag is set, the IP of the instruction with the undefined opcode is pushed onto the system stack, but the NMI trap is executed. After return from the NMI service routine, the IP is popped from the stack and immediately pushed again because of the pending UNDOPC trap.

External NMI Trap

Whenever a high to low transition on the dedicated external $\overline{\text{NMI}}$ pin (Non-Maskable Interrupt) is detected, the NMI flag in register TFR is set and the CPU will enter the NMI trap routine. The IP value pushed on the system stack is the address of the instruction following the one after which normal processing was interrupted by the NMI trap.

Note: The $\overline{\text{NMI}}$ pin is sampled with every CPU clock cycle to detect transitions.

Stack Overflow Trap

Whenever the stack pointer is decremented to a value which is less than the value in the stack overflow register STKOV, the STKOF flag in register TFR is set and the CPU will enter the stack overflow trap routine. Which IP value will be pushed onto the system stack depends on which operation caused the decrement of the SP. When an implicit decrement of the SP is made through a PUSH or CALL instruction, or upon interrupt or trap entry, the IP value pushed is the address of the following instruction. When the SP is decremented by a subtract instruction, the IP value pushed represents the address of the instruction after the instruction following the subtract instruction.

For recovery from stack overflow it must be ensured that there is enough excess space on the stack for saving the current system state (PSW, IP, in segmented mode also CSP) twice. Otherwise, a system reset should be generated.

Stack Underflow Trap

Whenever the stack pointer is incremented to a value which is greater than the value in the stack underflow register STKUN, the STKUF flag is set in register TFR and the CPU will enter the stack underflow trap routine. Again, which IP value will be pushed onto the system stack depends on which operation caused the increment of the SP. When an implicit increment of the SP is made through a POP or return instruction, the IP value pushed is the address of the following instruction. When the SP is incremented by an add instruction, the pushed IP value represents the address of the instruction after the instruction following the add instruction.

Undefined Opcode Trap

When the instruction currently decoded by the CPU does not contain a valid C164 opcode, the UNDOPC flag is set in register TFR and the CPU enters the undefined opcode trap routine. The IP value pushed onto the system stack is the address of the instruction that caused the trap.

This can be used to emulate unimplemented instructions. The trap service routine can examine the faulting instruction to decode operands for unimplemented opcodes based on the stacked IP. In order to resume processing, the stacked IP value must be incremented by the size of the undefined instruction, which is determined by the user, before a RETI instruction is executed.

Protection Fault Trap

Whenever one of the special protected instructions is executed where the opcode of that instruction is not repeated twice in the second word of the instruction and the byte following the opcode is not the complement of the opcode, the PRTFLT flag in register TFR is set and the CPU enters the protection fault trap routine. The protected instructions include DISWDT, EINIT, IDLE, PWRDN, SRST, and SRVWDT. The IP value pushed onto the system stack for the protection fault trap is the address of the instruction that caused the trap.

Illegal Word Operand Access Trap

Whenever a word operand read or write access is attempted to an odd byte address, the ILLOPA flag in register TFR is set and the CPU enters the illegal word operand access trap routine. The IP value pushed onto the system stack is the address of the instruction following the one which caused the trap.

Illegal Instruction Access Trap

Whenever a branch is made to an odd byte address, the ILLINA flag in register TFR is set and the CPU enters the illegal instruction access trap routine. The IP value pushed onto the system stack is the illegal odd target address of the branch instruction.

Illegal External Bus Access Trap

Whenever the CPU requests an external instruction fetch, data read or data write, and no external bus configuration has been specified, the ILLBUS flag in register TFR is set and the CPU enters the illegal bus access trap routine. The IP value pushed onto the system stack is the address of the instruction following the one which caused the trap.

6 Clock Generation

All activities of the C164's controller hardware and its on-chip peripherals are controlled via the system clock signal f_{CPU}.
This reference clock is generated in three stages (see also figure below):

- **Oscillator**

The on-chip Pierce oscillator can either run with an external crystal and appropriate oscillator circuitry or it can be driven by an external oscillator.

- **Frequency Control**

The input clock signal feeds the controller hardware...
...directly, providing phase coupled operation on not too high input frequency
...divided by 2 in order to get 50% duty cycle clock signal
...via an on-chip phase locked loop (PLL) providing maximum performance on low input frequency
...via the Slow Down Divider (SDD) in order to reduce the power consumption.
The resulting internal clock signal is referred to as "CPU clock" f_{CPU}.

- **Clock Drivers**

The CPU clock is distributed via separate clock drivers which feed the CPU itself and two groups of peripheral modules. The RTC is fed with the prescaled oscillator clock (f_{RTC}) via a separate clock driver, so it is not affected by the clock control functions.

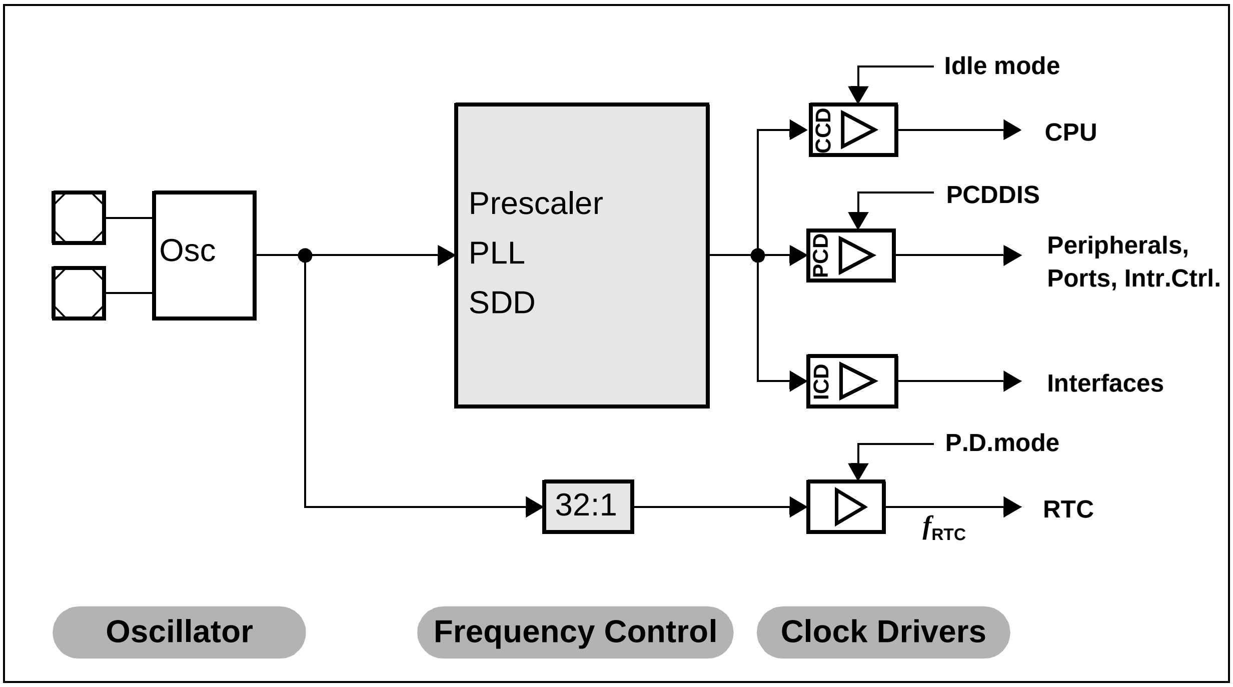


Figure 6-1 CPU Clock Generation Stages

6.1 Oscillator

The main oscillator of the C164 is a power optimized Pierce oscillator providing an inverter and a feedback element. Pins XTAL1 and XTAL2 connect the inverter to the external crystal. The standard external oscillator circuitry (see figure below) comprises the crystal, two low end capacitors and series resistor (Rx2) to limit the current through the crystal. The additional LC combination is only required for 3rd overtone crystals to suppress oscillation in the fundamental mode. A test resistor (R_Q) may be temporarily inserted to measure the oscillation allowance of the oscillator circuitry.

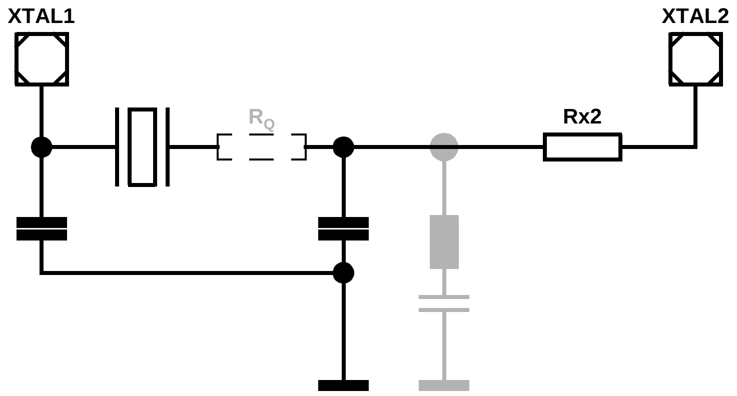


Figure 6-2 External Oscillator Circuitry

The on-chip oscillator is optimized for an input frequency range of 1 to 16 MHz.

An external clock signal (e.g. from an external oscillator or from a master device) may be fed to the input XTAL1. The Pierce oscillator then is not required to support the oscillation itself but is rather driven by the input signal. In this case the input frequency range may be 0 to 50 MHz (please note that the maximum applicable input frequency is limited by the device’s maximum CPU frequency).

For input frequencies above 25...30 MHz the oscillator’s output should be terminated as shown in the figure below, at lower frequencies it may be left open. This termination improves the operation of the oscillator by filtering out frequencies above the intended oscillator frequency.

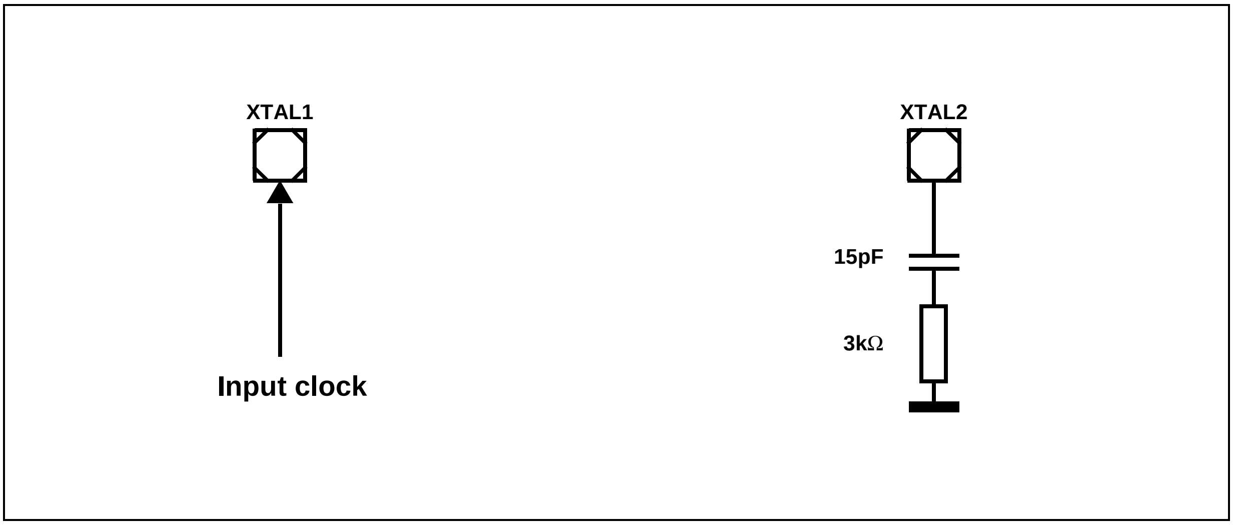


Figure 6-3 Oscillator Output Termination

Note: It is strongly recommended to measure the oscillation allowance (or margin) in the final target system (layout) to determine the optimum parameters for the oscillator operation.

6.2 Frequency Control

The CPU clock is generated from the oscillator clock in either of two software selectable ways:

The basic clock is the standard operating clock for the C164 and is required to deliver the intended maximum performance. The configuration via PORT0 (CLKCFG) after a long hardware reset determines one of three possible basic clock generation modes:

- Direct Drive: the oscillator clock is directly fed to the controller hardware.
- Prescaler: the oscillator clock is divided by 2 to achieve a 50% duty cycle.
- PLL: the oscillator clock is multiplied by a configurable factor of **F** = 1.5...5.

The Slow Down clock is the oscillator clock divided by a programmable factor of 1...32 (additional 2:1 divider in prescaler mode). This alternate possibility runs the C164 at a lower frequency (depending on the programmed slow down factor) and thus greatly reduces its power consumption.

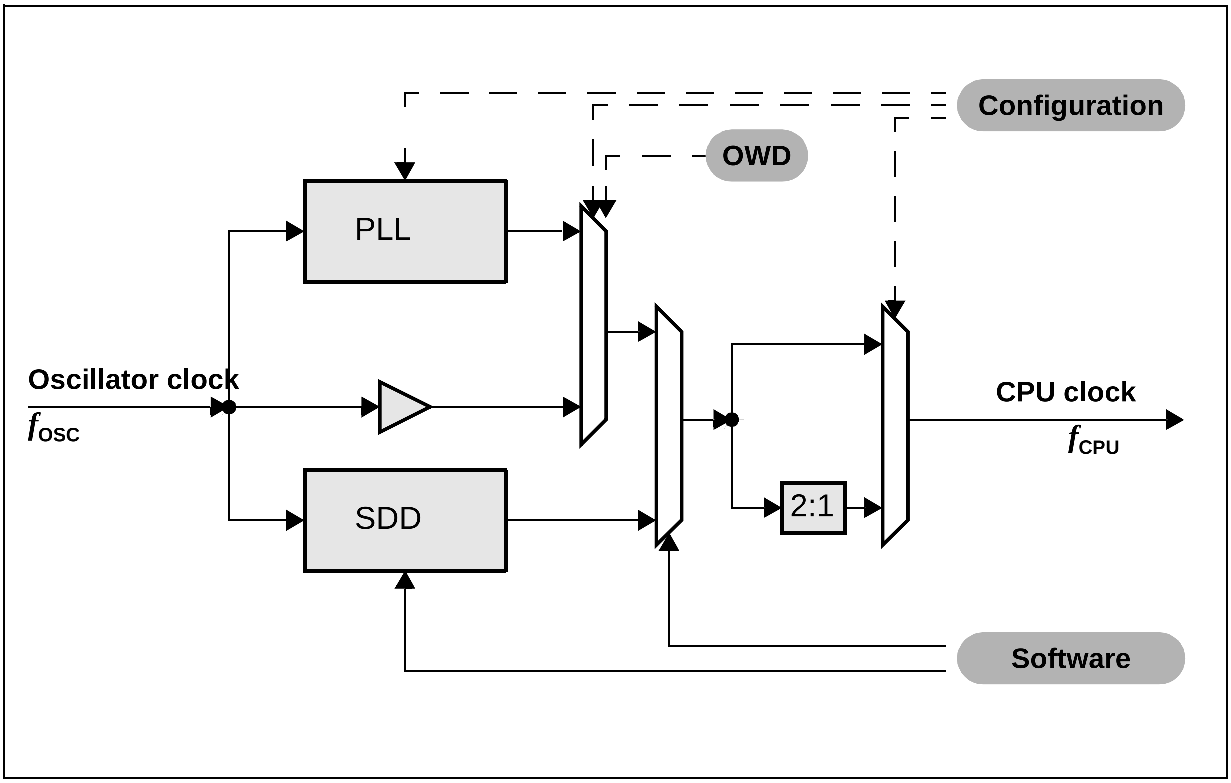


Figure 6-4 Frequency Control Paths

The internal operation of the C164 is controlled by the internal CPU clock f_{CPU}. Both edges of the CPU clock can trigger internal (e.g. pipeline) or external (e.g. bus cycles) operations (see figure below).

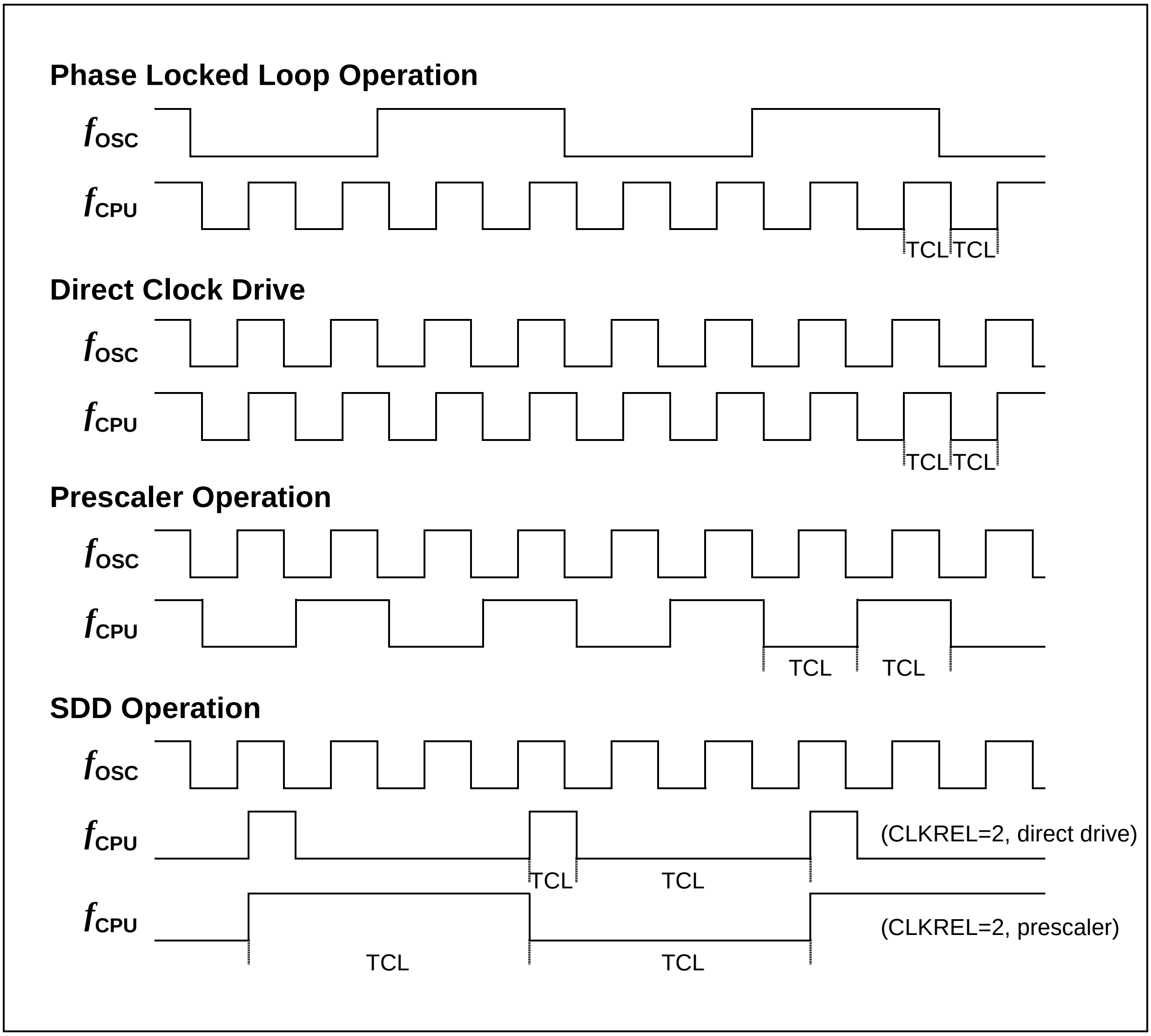


Figure 6-5 Generation Mechanisms for the CPU Clock

Direct Drive

When direct drive is configured (CLKCFG=’011’) the C164’s clock system is directly fed from the external clock input, i.e. $f_{CPU} = f_{OSC}$. This allows operation of the C164 with a reasonably small fundamental mode crystal. The specified minimum values for the CPU clock phases (TCLs) must be respected. Therefore the maximum input clock frequency depends on the clock signal’s duty cycle.

Prescaler Operation

When prescaler operation is configured (CLKCFG=’001’) the C164’s input clock is divided by 2 to generate then CPU clock signal, i.e. $f_{CPU} = f_{OSC}/2$. This requires the oscillator (or input clock) to run on 2 times the intended operating frequency but guarantees a 50% duty cycle for the internal clock system independent of the input clock signal’s waveform.

PLL Operation

When PLL operation is configured (via CLKCFG) the C164’s input clock is fed to the on-chip phase locked loop circuit which multiplies its frequency by a factor of **F** = 1.5...5 (selectable via CLKCFG, see table below) and generates a CPU clock signal with 50% duty cycle, i.e. $f_{CPU} = f_{OSC}$*F.

The on-chip PLL circuit allows operation of the C164 on a low frequency external clock while still providing maximum performance. The PLL also provides fail safe mechanisms which allow the detection of frequency deviations and the execution of emergency actions in case of an external clock failure.

When the PLL detects a missing input clock signal it generates an interrupt request. This warning interrupt indicates that the PLL frequency is no more locked, i.e. no more stable. This occurs when the input clock is unstable and especially when the input clock fails completely, e.g. due to a broken crystal. In this case the synchronization mechanism will reduce the PLL output frequency down to the PLL’s base frequency (2...5 MHz). The base frequency is still generated and allows the CPU to execute emergency actions in case of a loss of the external clock.

On power-up the PLL provides a stable clock signal within ca. 1 ms after V_{DD} has reached the specified valid range, even if there is no external clock signal (in this case the PLL will run on its base frequency of 2...5 MHz). The PLL starts synchronizing with the external clock signal as soon as it is available. Within ca. 1 ms after stable oscillations of the external clock within the specified frequency range the PLL will be synchronous with this clock at a frequency of **F** * f_{OSC}, i.e. the PLL locks to the external clock.

When PLL operation is selected the CPU clock is a selectable multiple of the oscillator frequency, i.e. the input frequency.

The table below lists the possible selections.

Table 6-1 C164 Clock Generation Modes

P0.15-13 (P0H.7-5)	CPU Frequency $f_{CPU} = f_{OSC} * F$	External Clock Input Range [1]	Notes
1 1 1	f_{OSC} * 4	2.5 to 6.25 MHz	Default configuration
1 1 0	f_{OSC} * 3	3.33 to 8.33 MHz	
1 0 1	f_{OSC} * 2	5 to 12.5 MHz	
1 0 0	f_{OSC} * 5	2 to 5 MHz	
0 1 1	f_{OSC} * 1	1 to 25 MHz	Direct drive [2]
0 1 0	f_{OSC} * 1.5	6.66 to 16.6 MHz	
0 0 1	f_{OSC} / 2	2 to 50 MHz	CPU clock via prescaler
0 0 0	f_{OSC} * 2.5	4 to 10 MHz	

[1] The external clock input range refers to a CPU clock range of 10...25 MHz.

[2] The maximum frequency depends on the duty cycle of the external clock signal. In emulation mode pin P0.15 (P0H.7) is inverted, i.e. the configuration ’111’ would select direct drive in emulation mode.

The PLL constantly synchronizes to the external clock signal. Due to the fact that the external frequency is 1/**F**’th of the PLL output frequency the output frequency may be slightly higher or lower than the desired frequency. This jitter is irrelevant for longer time periods. For short periods (1...4 CPU clock cycles) it remains below 4%.

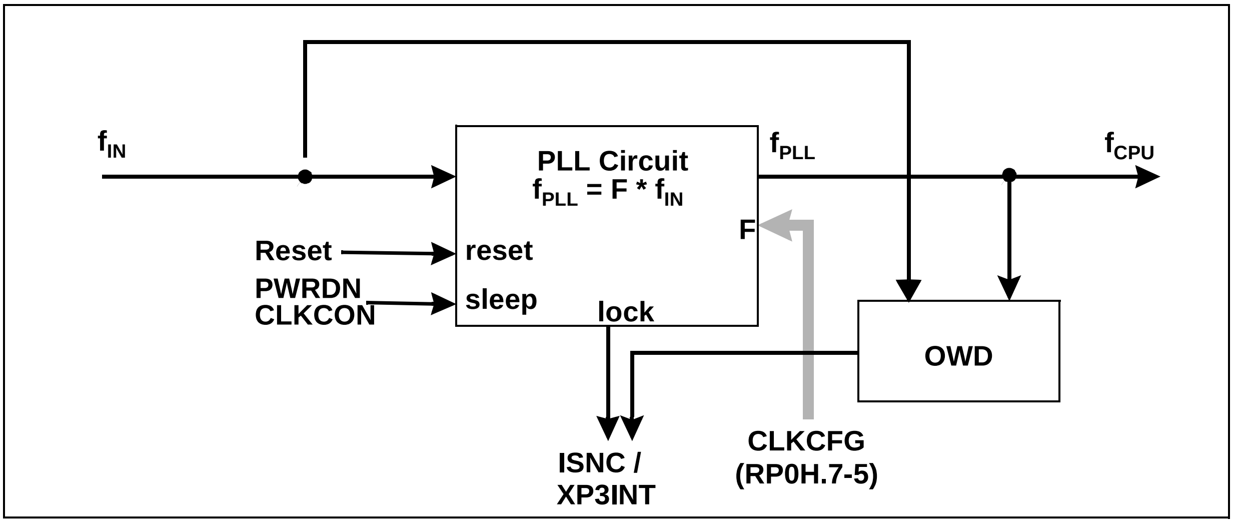


Figure 6-6 PLL Block Diagram

6.3 Oscillator Watchdog

The C164 provides an Oscillator Watchdog (OWD) which monitors the clock signal fed to input XTAL1 of the on-chip oscillator (either with a crystal or via external clock drive) in prescaler or direct drive mode (not if the PLL provides the basic clock). For this operation the PLL provides a clock signal (base frequency) which is used to supervise transitions on the oscillator clock. This PLL clock is independent from the XTAL1 clock. When the expected oscillator clock transitions are missing the OWD activates the PLL Unlock / OWD interrupt node and supplies the CPU with the PLL clock signal instead of the selected oscillator clock. Under these circumstances the PLL will oscillate with its base frequency.

In direct drive mode the PLL base frequency is used directly (f_{CPU} = 2...5 MHz).
In prescaler mode the PLL base frequency is divided by 2 (f_{CPU} = 1...2.5 MHz).

If the oscillator clock fails while the PLL provides the basic clock the system will be supplied with the PLL base frequency anyway.

With this PLL clock signal the CPU can either execute a controlled shutdown sequence bringing the system into a defined and safe idle state, or it can provide an emergency operation of the system with reduced performance based on this (normally slower) emergency clock.

Note: The CPU clock source is only switched back to the oscillator clock after a hardware reset.

The oscillator watchdog can be disabled by setting bit OWDDIS in register SYSCON. In this case the PLL remains idle and provides no clock signal, while the CPU clock signal is derived directly from the oscillator clock or via prescaler or SDD. Also no interrupt request will be generated in case of a missing oscillator clock.

Note: At the end of an external reset bit OWDDIS reflects the inverted level of pin $\overline{RD}$ at that time. Thus the oscillator watchdog may also be disabled via hardware by (externally) pulling the $\overline{RD}$ line low upon a reset, similar to the standard reset configuration via PORT0.

The oscillator watchdog cannot provide full security while the CPU clock signal is generated by the SlowDown Divider, because the OWD cannot switch to the PLL clock in this case (see Figure 6-4 on page 4). OWD interrupts are only recognizable if f_{OSC} is still available (e.g. input frequency too low or intermittent failure only).
A broken crystal cannot be detected by software (OWD interrupt server) as no SDD clock is available in such a case.

6.4 Clock Drivers

The operating clock signal f_{CPU} is distributed to the controller hardware via several clock drivers which are disabled under certain circumstances. The real time clock RTC is clocked via a separate clock driver which delivers the prescaled oscillator clock (contrary to the other clock drivers). The table below summarizes the different clock drivers and their function, especially in power reduction modes:

Table 6-2 Clock Drivers Description

Clock Driver	**Clock Signal**	**Active mode**	**Idle mode**	**Power Down and Sleep mode**	**Connected Circuitry**
CCD CPU Clock Driver	f_{CPU}	ON	Off	Off	CPU, internal memory modules (IRAM, ROM/OTP/Flash)
ICD Interface Clock Driver	f_{CPU}	ON	ON	Off	ASC0, WDT, SSC, interrupt detection circuitry
PCD Peripheral Clock Driver	f_{CPU}	Control via PCDDIS	Control via PCDDIS	Off	(X)Peripherals (timers, etc.) except those driven by ICD, interrupt controller, ports
RCD RTC Clock Driver	f_{RTC}	ON	ON	Control via PDCON / SLEEP-CON	Realtime clock

Note: Disabling PCD by setting bit PCDDIS stops the clock signal for all connected modules. Make sure that all these modules are in a safe state before stopping their clock signal.
The port input and output values will not change while PCD is disabled (ASC0 and SSC will still operate, if active),
CLKOUT will be high if enabled.
Please also respect the hints given in section „Flexible Peripheral Management“ of chapter „Power Management“.

7 Parallel Ports

In order to accept or generate single external control signals or parallel data, the C164 provides up to 59 parallel IO lines organized into four 8-bit IO ports (PORT0 made of P0H and P0L, PORT1 made of P1H and P1L), one 9-bit IO port (Port 3), one 6-bit IO port (Port 4), one 4-bit IO port (Port 8), and one 8-bit input port (Port 5).

These port lines may be used for general purpose Input/Output controlled via software or may be used implicitly by the C164's integrated peripherals or the External Bus Controller.

All port lines are bit addressable, and all input/output lines are individually (bit-wise) programmable as inputs or outputs via direction registers (except Port 5, of course). The IO ports are true bidirectional ports which are switched to high impedance state when configured as inputs. The output drivers of three IO ports (3, 4, 8) can be configured (pin by pin) for push/pull operation or open-drain operation via control output.

The logic level of a pin is clocked into the input latch once per state time, regardless whether the port is configured for input or output.

A write operation to a port pin configured as an input causes the value to be written into the port output latch, while a read operation returns the latched state of the pin itself. A read-modify-write operation reads the value of the pin, modifies it, and writes it back to the output latch.

Writing to a pin configured as an output (DPx.y='1') causes the output latch and the pin to have the written value, since the output buffer is enabled. Reading this pin returns the value of the output latch. A read-modify-write operation reads the value of the output latch, modifies it, and writes it back to the output latch, thus also modifying the level at the pin.

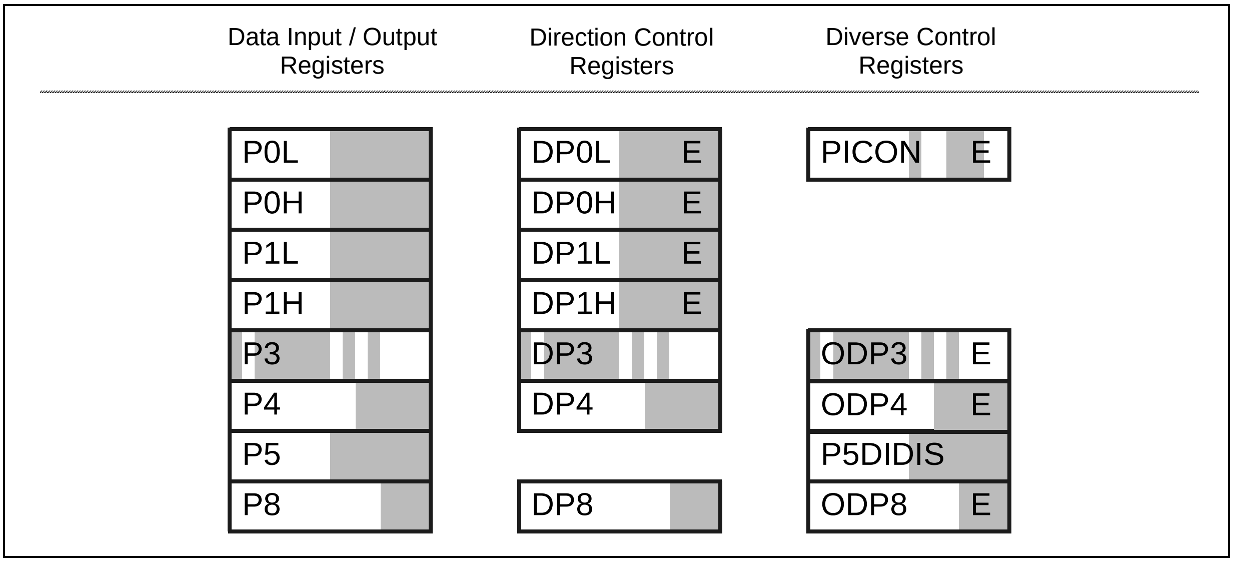


Figure 7-1 SFRs and Pins associated with the Parallel Ports

7.1 Input Threshold Control

The standard inputs of the C164 determine the status of input signals according to TTL levels. In order to accept and recognize noisy signals, CMOS-like input thresholds can be selected instead of the standard TTL thresholds for all pins of specific ports. These special thresholds are defined above the TTL thresholds and feature a defined hysteresis to prevent the inputs from toggling while the respective input signal level is near the thresholds.

The Port Input Control register PICON allows to select these thresholds for each byte of the indicated ports, i.e. 8-bit ports are controlled by one bit each while 16-bit ports are controlled by two bits each.

PICON

Port Input Control Register ESFR ($F1C4_H$/$E2_H$) Reset value: - - 00_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P8 LIN	**-**	**-**	**P4 LIN**	**P3 HIN**	**P3 LIN**	**-**	**-**
-	-	-	-	-	-	-	-	rw	-	-	rw	rw	rw	-	-

Bit	Function
PxLIN	**Port x Low Byte Input Level Selection** 0: Pins Px[7-0] switch on standard TTL input levels 1: Pins Px[7-0] switch on special threshold input levels
PxHIN	**Port x High Byte Input Level Selection** 0: Pins Px[15-8] switch on standard TTL input levels 1: Pins Px[15-8] switch on special threshold input levels

All options for individual direction and output mode control are available for each pin independent from the selected input threshold.

The input hysteresis provides stable inputs from noisy or slowly changing external signals.

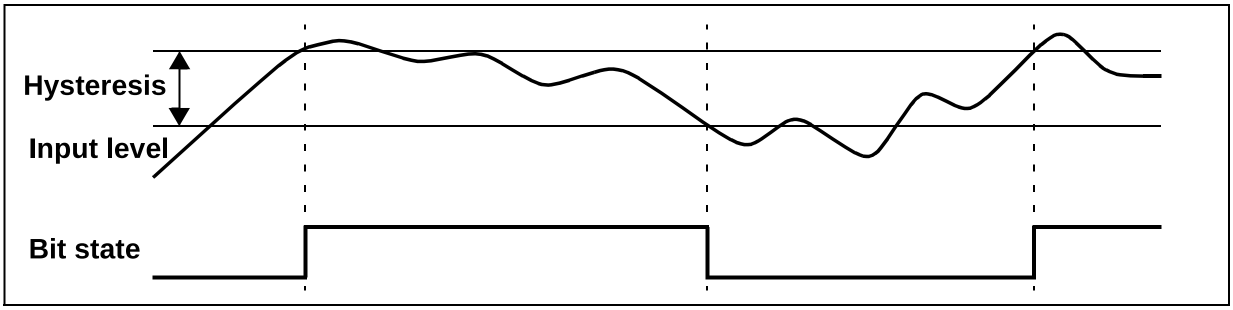


Figure 7-2 Hysteresis for Special Input Thresholds

7.2 Output Driver Control

The output driver of a port pin is activated by switching the respective pin to output, i.e. DPx.y = ’1’. The value that is driven to the pin is determined by the port output latch or by the associated alternate function (e.g. address, peripheral IO, etc.). The user software can control the characteristics of the output driver via the following mechanisms:

- **Open Drain Mode**: The upper (push) transistor is always disabled. Only ’0’ is driven actively, an external pullup is required.
- **Driver Characteristic**: The driver strength (static and dynamic behaviour) can be selected.
- **Edge Characteristic**: The rise/fall time of an output signal can be selected.

Open Drain Mode

In the C164 certain ports provide Open Drain Control, which allows to switch the output driver of a port pin from a push/pull configuration to an open drain configuration. In push/pull mode a port output driver has an upper and a lower transistor, thus it can actively drive the line either to a high or a low level. In open drain mode the upper transistor is always switched off, and the output driver can only actively drive the line to a low level. When writing a ‘1’ to the port latch, the lower transistor is switched off and the output enters a high-impedance state. The high level must then be provided by an external pullup device. With this feature, it is possible to connect several port pins together to a Wired-AND configuration, saving external glue logic and/or additional software overhead for enabling/disabling output signals.

This feature is controlled through the respective Open Drain Control Registers ODPx which are provided for each port that has this feature implemented. These registers allow the individual bit-wise selection of the open drain mode for each port line.

If the respective control bit ODPx.y is ‘0’ (default after reset), the output driver is in the push/pull mode. If ODPx.y is ‘1’, the open drain configuration is selected. Note that all ODPx registers are located in the ESFR space.

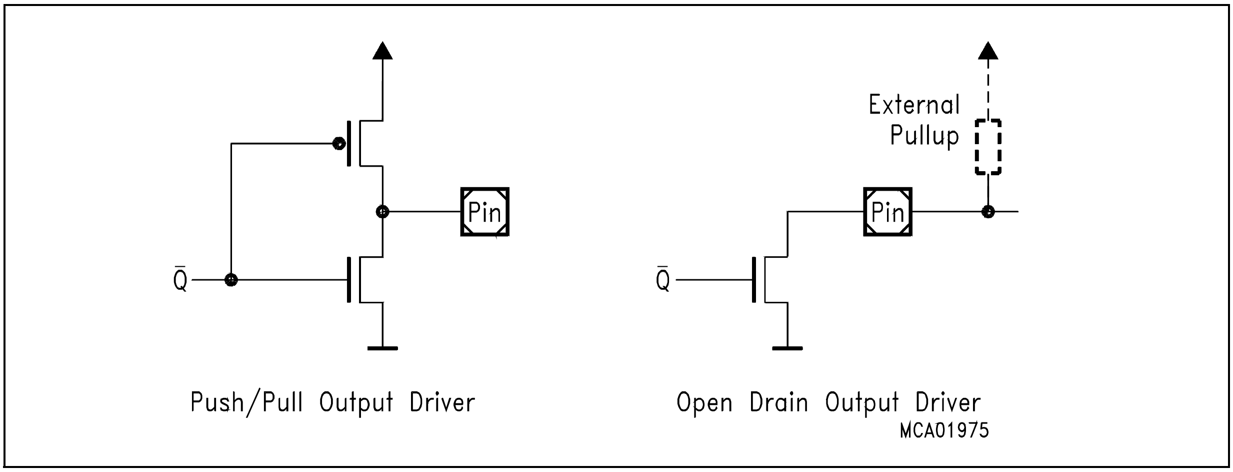


Figure 7-3 Output Drivers in Push/Pull Mode and in Open Drain Mode

Driver Characteristic

This defines either the general driving capability of the respective driver, or if the driver strength is reduced after the target output level has been reached or not. Reducing the driver strength increases the output's internal resistance which attenuates noise that is imported/exported via the output line. For driving LEDs or power transistors, however, a stable high output current may still be required.

The controllable output drivers of the C167CS pins feature two differently sized transistors (strong and weak) for each direction (push and pull). The time of activating/ deactivating these transistors determines the output characteristics of the respective port driver.

Three modes can be selected to adapt the driver characteristics to the application's requirements:

In High Current Mode both transistors are activated all the time. In this case the driver provides maximum output current even after the target signal level is reached.

In Low Noise Mode both transistors are activated at the beginning of a signal transition. When the target signal level is reached the driver strength is reduced by switching off the strong transistor. The weak transistor will keep the specified output level while the susceptibility for noise is reduced.

In Low Current Mode only the weak transistor is activated while the strong transistor remains off. This results in smooth transitions with low current peaks (and reduced susceptibility for noise) on the cost of increased transition times, i.e. slower edges, depending on the capacitive load.

Edge Characteristic

This defines the rise/fall time for the respective output, i.e. the output transition time. Slow edges reduce the peak currents that are drawn when changing the voltage level of an external capacitive load. For a bus interface, however, fast edges may still be required. Edge characteristic effects the pre-driver which controls the final output driver stage.

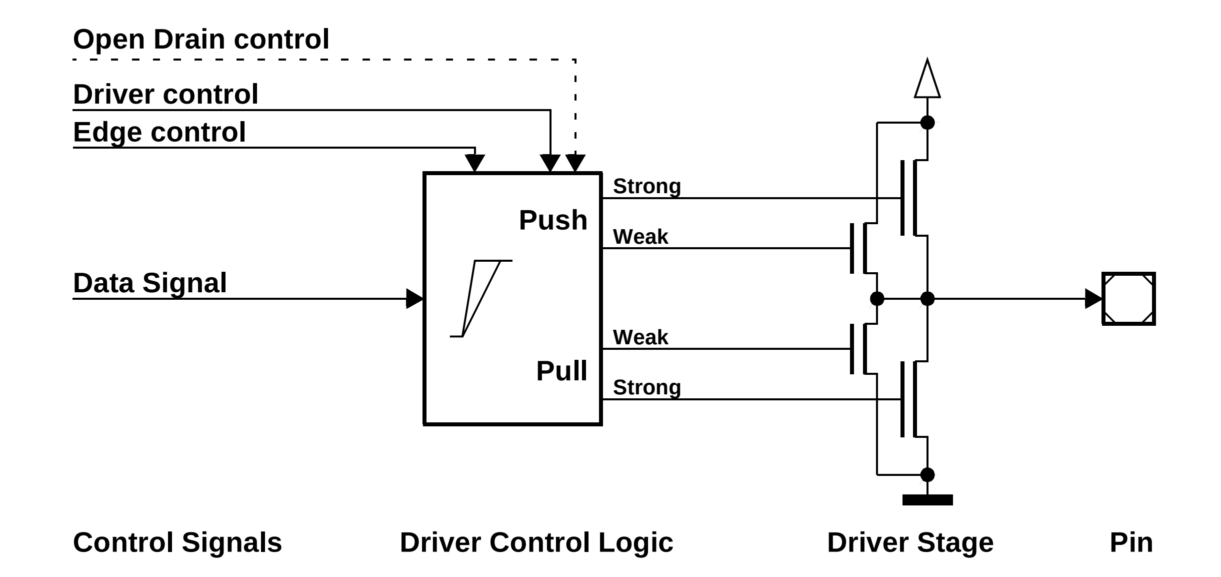


Figure 7-4 Structure of Two-Level Output Driver with Edge Control

Table 7-1 Output Transistor Operation

Driver Mode		Low Current Mode		Dynamic Current Mode		High Current Mode	
Output Level		**’0’**	**’1'**	**’0’**	**’1'**	**’0’**	**’1'**
Push [1)]transistors	Strong	---	---	---	⎍	---	ON
	Weak	---	ON	---	ON	---	ON
Pull transistors	Strong	---	---	⎍	---	ON	---
	Weak	ON	---	ON	---	ON	---

1) The upper (push) transistors are always off for output pins that operate in open drain mode.

The **Port Output Control registers** POCONx provide the corresponding control bits. For each feature (edge/driver characteristic and for each port nibble) a 2-bit control field is provided (i.e. 4 bits for each port nibble). Word ports consume four control nibbles each, byte ports consume two control nibbles each, where each control nibble controls 4 pins of the respective port.

The general register layout shown below is valid for all POCON registers. Please note that for byte ports only two pairs of bitfields are provided (see register allocation table).

POCON*

Port * Output Ctrl. Reg. **ESFR (F0xx$_H$/yy$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PN3DC		PN3EC		PN2DC		PN2EC		PN1DC		PN1EC		PN0DC		PN0EC	
rw		rw		rw		rw		rw		rw		rw		rw	

Bit	Function
PNxEC	**Port Nibble x Edge Characteristic** (Defines the output rise/fall time t_{RF}) 00: Fast edge mode, rise/fall times depend on the driver's dimensioning. 01: Reduced edge mode. 10: *Reserved*. 11: *Reserved*.
PNxDC	**Port Nibble x Driver Characteristic** (Defines the delivered output current) 00: High Current mode: Driver always operates with maximum strength. 01: Low Current mode: Driver always operates with reduced strength. 10: Dynamic Current mode: Driver strength is reduced after the target level has been reached. 11: *Reserved*.

The table below lists the defined POCON registers and the allocation of control bitfields and port pins.

Table 7-2 Port Output Control Register Allocation

Control Register	Location	Controlled Port				Notes
POCON20	F0AA$_H$ / 55$_H$	$\overline{\text{RSTOUT}}$	CLKOUT/ FOUT	ALE,	$\overline{\text{WR}}$, $\overline{\text{RD}}$ $\overline{\text{BHE}}$/WH	No associated port
POCON8	F092$_H$ / 49$_H$			---	P8.3-0	
POCON4	F08C$_H$ / 46$_H$			P4.6-5,	P4.3-0	P4.7, P4.4 missing
POCON3	F08A$_H$ / 45$_H$	P3.15-12,	P3.11-8,	P3.7-4	---	P3.14, P3.7, P3.5 missing
POCON1H	F086$_H$ / 43$_H$			P1H.7-4,	P1H.3-0	
POCON1L	F084$_H$ / 42$_H$			P1L.7-4,	P1L.3-0	
POCON0H	F082$_H$ / 41$_H$			P0H.7-4,	P0H.3-0	
POCON0L	F080$_H$ / 40$_H$			P0L.7-4,	P0L.3-0	

The figure below summarizes the effects of the driver characteristics:
Edge characteristic generally influences the output signal's **shape**. Driver characteristic influences the signal shape's susceptibility to the external **capacitive load**.

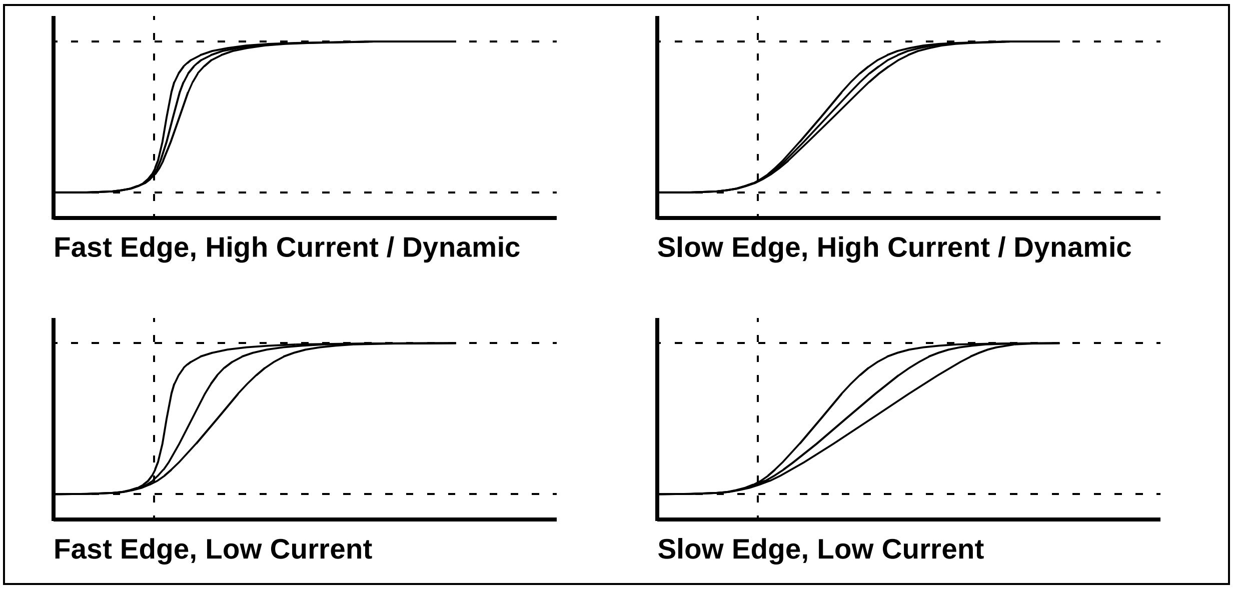


Figure 7-5 General Output Signal Waveforms

Port Driver Temperature Compensation

The temperature compensation for the port drivers provides driver output characteristics which are stable (within a certain band of parameter variation) over the specified temperature range, e.g. -40...+125°C.

Generally, temperature compensation is a transparent feature. Still register PTCR provides access to the actual compensation value and even allows software control of this mechanism.

This is useful in two cases:

- **Device testing**: the function of the compensation mechanism can be verified during production testing or characterization.
- **User control**: during operation the device can be controlled via externally provided compensation values rather than via the internal mechanism.

PTCR
Port Temp. Comp. Reg. **ESFR (F0AE$_H$/57$_H$)** **Reset value: 000X$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	TCS	-	TCC		TCD	-	TCV	
-	-	-	-	-	-	-	-	rw	-	rw		rw	-	rwh	

Bit	Function
TCV	**Temperature Compensation Value** The value which is currently generated by the temperature compensation sensor. This value is fed to the port logic while bit TCS = '0'. *Note: Bitfield TCV is not affected by a reset, but rather indicates the current sensor value at any time.*
TCD	**Temperature Compensation Disable** 0: The temperature compensation is active. 1: The temperature compensation sensor is deactivated. The port logic is controlled by the most recent sensor value.
TCC	**Temperature Compensation Control** This value is fed to the port logic instead of the temperature compensation sensor value, while bit TCS = '1'.
TCS	**Temperature Compensation Source** 0: Port logic is controlled by the temperature compensation sensor. 1: Port logic is controlled by software via bitfield TCC.

7.3 Alternate Port Functions

In order to provide a maximum of flexibility for different applications and their specific IO requirements, port lines have programmable alternate input or output functions associated with them.

Table 7-3 Summary of Alternate Port Functions

Port	Alternate Function(s)	Alternate Signal(s)
PORT0	Address and data lines when accessing external resources (e.g. memory)	AD15 ... AD0
PORT1	Address lines when accessing external resources (e.g. memory) Capture inputs or compare outputs of the CAPCOM units Fast external interrupt inputs CAPCOM timer input	A15 ... A0, CC27IO ... CC24IO, $\overline{\text{CTRAP}}$, CC6n, COUT6n, $\overline{\text{CC6POSn}}$, EX3IN ... EX0IN, T7IN
Port 3	System clock or programmable frequ. output Optional bus control signal Input/output functions of serial interfaces, timers	CLKOUT/FOUT, $\overline{\text{BHE}}$/$\overline{\text{WRH}}$, RxD0, TxD0, MTSR, MRST, SCLK, T3IN, T3EUD
Port 4	Selected segment address lines in systems with more than 64 KBytes of external resources Optional chip select output signals CAN interface (when assigned)	A21 ... A16, $\overline{\text{CS3}}$ ... $\overline{\text{CS0}}$, CAN1_TxD, CAN1_RxD
Port 5	Analog input channels to the A/D converter Timer control signal inputs	AN7 ... AN0, T2EUD, T4EUD, T2IN, T4IN
Port 8	Capture inputs or compare outputs of the CAPCOM2 unit CAN interface (when assigned)	CC19IO ... CC16IO, CAN1_TxD, CAN1_RxD

If an **alternate output function** of a pin is to be used, the direction of this pin must be programmed for output (DPx.y=‘1’), except for some signals that are used directly after reset and are configured automatically. Otherwise the pin remains in the high-impedance state and is not effected by the alternate output function. The respective port latch should hold a ‘1’, because its output is combined with the alternate output data.
X-Peripherals (peripherals connected to the on-chip XBUS) control their associated IO pins directly via separate control lines.

If an **alternate input function** of a pin is used, the direction of the pin must be programmed for input (DPx.y=‘0’) if an external device is driving the pin. The input direction is the default after reset. If no external device is connected to the pin, however, one can also set the direction for this pin to output. In this case, the pin reflects the state of the port output latch. Thus, the alternate input function reads the value stored in the port output latch. This can be used for testing purposes to allow a software trigger of an alternate input function by writing to the port output latch.

On most of the port lines, the user software is responsible for setting the proper direction when using an alternate input or output function of a pin. This is done by setting or clearing the direction control bit DPx.y of the pin before enabling the alternate function. There are port lines, however, where the direction of the port line is switched automatically. For instance, in the multiplexed external bus modes of PORT0, the direction must be switched several times for an instruction fetch in order to output the addresses and to input the data. Obviously, this cannot be done through instructions. In these cases, the direction of the port line is switched automatically by hardware if the alternate function of such a pin is enabled.
To determine the appropriate level of the port output latches check how the alternate data output is combined with the respective port latch output.

There is one basic structure for all port lines with only an alternate input function. Port lines with only an alternate output function, however, have different structures due to the way the direction of the pin is switched and depending on whether the pin is accessible by the user software or not in the alternate function mode.

All port lines that are not used for these alternate functions may be used as general purpose IO lines. When using port pins for general purpose output, the initial output value should be written to the port latch prior to enabling the output drivers, in order to avoid undesired transitions on the output pins. This applies to single pins as well as to pin groups (see examples below).

```
OUTPUT_ENABLE_SINGLE_PIN:
BSET        P4.0                   ;Initial output level is ’high’
BSET        DP4.0                  ;Switch on the output driver

OUTPUT_ENABLE_PIN_GROUP:
BFLDL       P4, #05H, #05H         ;Initial output level is ’high’
BFLDL       DP4, #05H, #05H        ;Switch on the output drivers
```

Note: When using several BSET pairs to control more pins of one port, these pairs must be separated by instructions, which do not reference the respective port (see “Particular Pipeline Effects” in chapter “The Central Processing Unit”).

Each of these ports and the alternate input and output functions are described in detail in the following subsections.

7.4 PORT0

The two 8-bit ports P0H and P0L represent the higher and lower part of PORT0, respectively. Both halfs of PORT0 can be written (e.g. via a PEC transfer) without effecting the other half.
If this port is used for general purpose IO, the direction of each line can be configured via the corresponding direction registers DP0H and DP0L.

P0L
PORT0 Low Register **SFR (FF00$_H$/80$_H$** **Reset value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P0L .7	**P0L .6**	**P0L .5**	**P0L .4**	**P0L .3**	**P0L .2**	**P0L .1**	**P0L .0**
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

P0H
PORT0 High Register **SFR (FF02$_H$/81$_H$)** **Reset value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P0H .7	**P0H .6**	**P0H .5**	**P0H .4**	**P0H .3**	**P0H .2**	**P0H .1**	**P0H .0**
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
P0X.y	**Port data register P0H or P0L bit y**

DP0L
P0L Direction Ctrl. Register **ESFR (F100$_H$/80$_H$)** **Reset value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								DP0L .7	**DP0L .6**	**DP0L .5**	**DP0L .4**	**DP0L .3**	**DP0L .2**	**DP0L .1**	**DP0L .0**
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

DP0H
P0H Direction Ctrl. Register **ESFR (F102$_H$/81$_H$)** **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								DP0H .7	**DP0H .6**	**DP0H .5**	**DP0H .4**	**DP0H .3**	**DP0H .2**	**DP0H .1**	**DP0H .0**
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
DP0X.y	**Port direction register DP0H or DP0L bit y** DP0X.y = 0: Port line P0X.y is an input (high-impedance) DP0X.y = 1: Port line P0X.y is an output

7.4.1 Alternate Functions of PORT0

When an external bus is enabled, PORT0 is used as address/data bus.
PORT0 is also used to select the system startup configuration. During reset, PORT0 is configured to input, and each line is held high through an internal pullup device. Each line can now be individually pulled to a low level (see DC-level specifications in the respective Data Sheets) through an external pulldown device. A default configuration is selected when the respective PORT0 lines are at a high level. Through pulling individual lines to a low level, this default can be changed according to the needs of the applications.
The internal pullup devices are designed such that an external pulldown resistors (see Data Sheet specification) can be used to apply a correct low level. These external pulldown resistors can remain connected to the PORT0 pins also during normal operation, however, care has to be taken such that they do not disturb the normal function of PORT0 (this might be the case, for example, if the external resistor is too strong).

With the end of reset, the selected bus configuration will be written to the BUSCON0 register. The configuration of the high byte of PORT0 will be copied into the special register RP0H. This read-only register holds the selection for the number of chip selects and segment addresses. Software can read this register in order to react according to the selected configuration, if required.
When the reset is terminated, the internal pullup devices are switched off, and PORT0 will be switched to the appropriate operating mode.

During external accesses in multiplexed bus modes PORT0 first outputs the 16-bit intra-segment address as an alternate output function. PORT0 is then switched to high-impedance input mode to read the incoming instruction or data. In 8-bit data bus mode, two memory cycles are required for word accesses, the first for the low byte and the second for the high byte of the word. During write cycles PORT0 outputs the data byte or word after outputting the address.

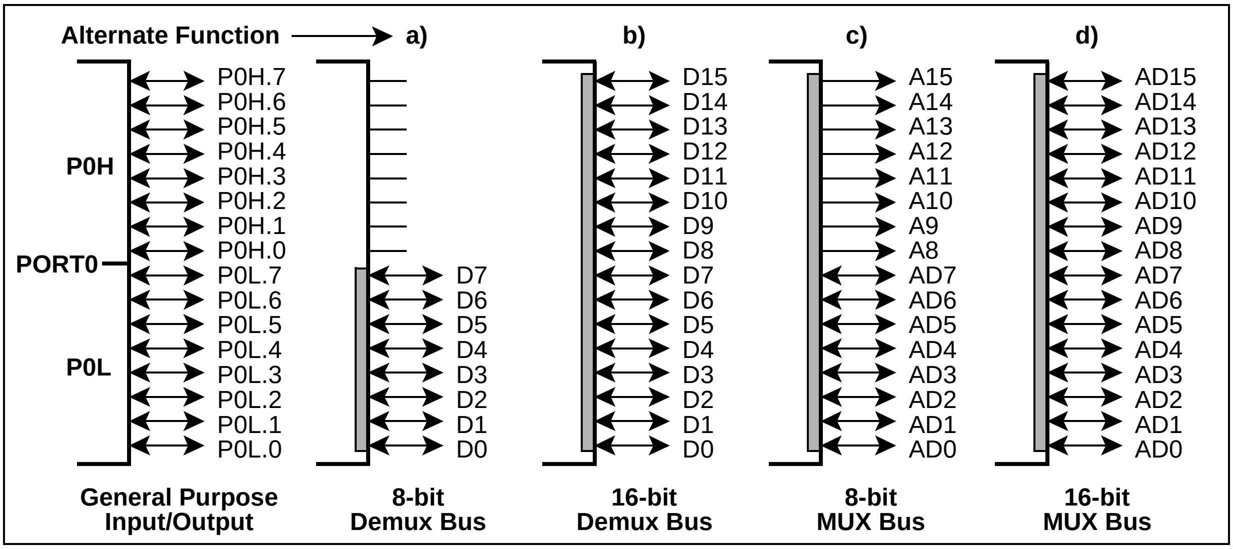


Figure 7-6 PORT0 IO and Alternate Functions

When an external bus mode is enabled, the direction of the port pin and the loading of data into the port output latch are controlled by the bus controller hardware. The input of the port output latch is disconnected from the internal bus and is switched to the line labeled “Alternate Data Output” via a multiplexer. The alternate data can be the 16-bit intrasegment address or the 8/16-bit data information. The incoming data on PORT0 is read on the line “Alternate Data Input”. While an external bus mode is enabled, the user software should not write to the port output latch, otherwise unpredictable results may occur. When the external bus modes are disabled, the contents of the direction register last written by the user becomes active.

The figure below shows the structure of a PORT0 pin.

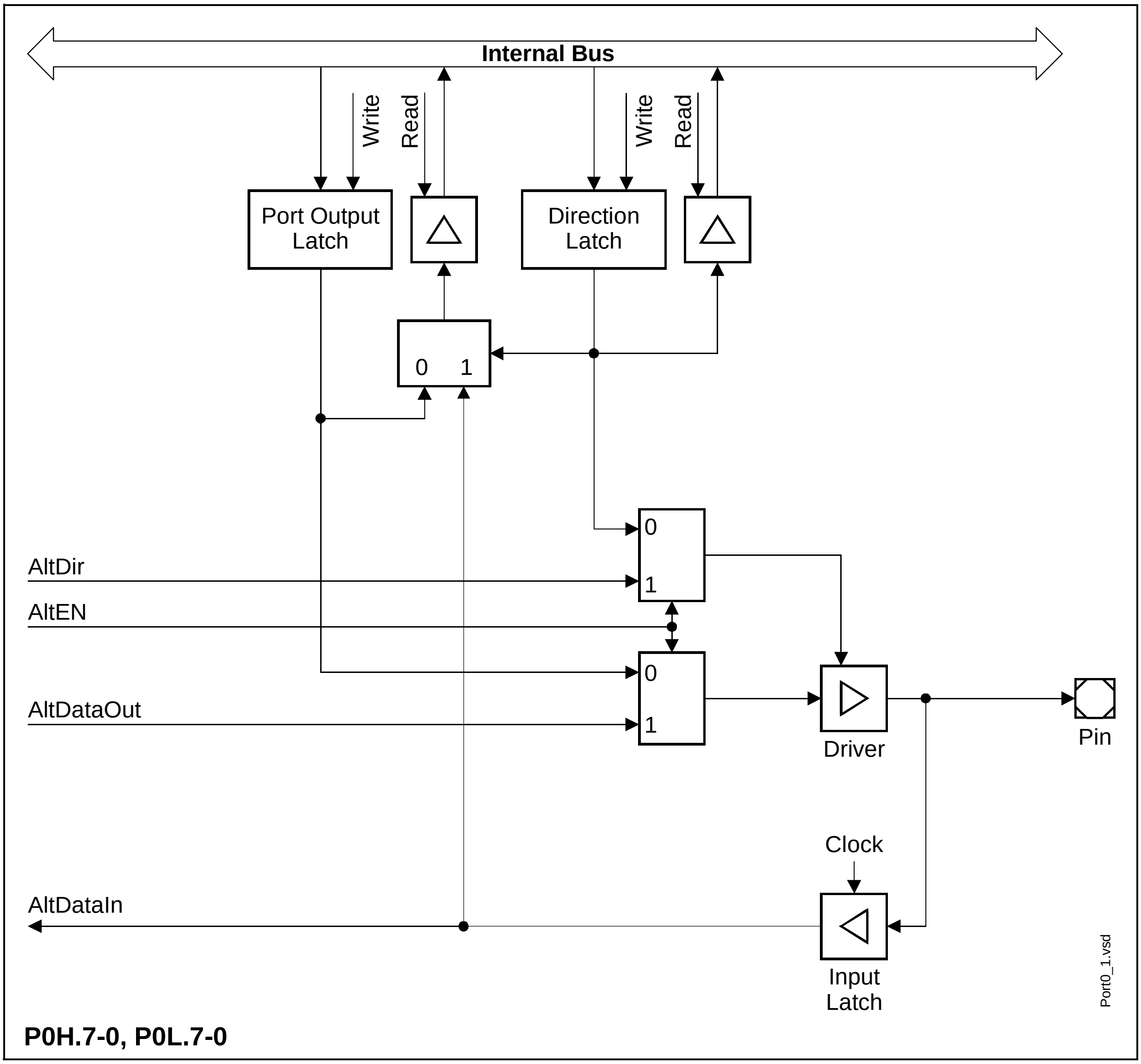


Figure 7-7 Block Diagram of a PORT0 Pin

7.5 PORT1

The two 8-bit ports P1H and P1L represent the higher and lower part of PORT1, respectively. Both halfs of PORT1 can be written (e.g. via a PEC transfer) without effecting the other half.
If this port is used for general purpose IO, the direction of each line can be configured via the corresponding direction registers DP1H and DP1L.

P1L
PORT1 Low Register **SFR (FF04$_H$/82$_H$)** **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P1L .7	**P1L 6**	**P1L .5**	**P1L .4**	**P1L .3**	**P1L .2**	**P1L .1**	**P1L .0**
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

P1H
PORT1 High Register **SFR (FF06$_H$/83$_H$)** **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P1H .7	**P1H .6**	**P1H .5**	**P1H .4**	**P1H .3**	**P1H .2**	**P1H .1**	**P1H .0**
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
P1X.y	**Port data register P1H or P1L bit y**

DP1L
P1L Direction Ctrl. Register **ESFR (F104$_H$/82$_H$)** **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								DP1 L.7	**DP1 L.6**	**DP1 L.5**	**DP1 L.4**	**DP1 L.3**	**DP1 L.2**	**DP1 L.1**	**DP1 L.0**
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

DP1H
P1H Direction Ctrl. Register **ESFR (F106$_H$/83$_H$)** **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								DP1 H.7	**DP1 H.6**	**DP1 H.5**	**DP1 H.4**	**DP1 H.3**	**DP1 H.2**	**DP1 H.1**	**DP1 H.0**
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
DP1X.y	**Port direction register DP1H or DP1L bit y** DP1X.y = 0: Port line P1X.y is an input (high-impedance) DP1X.y = 1: Port line P1X.y is an output

Alternate Functions of PORT1

When a demultiplexed external bus is enabled, PORT1 is used as address bus.
Note that demultiplexed bus modes use PORT1 as a 16-bit port. Otherwise all 16 port lines can be used for general purpose IO.

The lower 11 pins of PORT1 (P1H.2...P1L.0) serve as the inputs/outputs for the CAPCOM6 unit.

Pins P1H.3...P1H.0 accept the fast external inputs. P1H.3 also serves as input for timer T7.

The upper four pins of PORT1 (P1H.7...P1H.4) also serve as capture inputs or compare outputs for the CAPCOM2 unit (CC27IO...CC24IO).
As all other capture inputs, the capture input function of pins P1H.7...P1H.4 can also be used as external interrupt inputs (sample rate 16 TCL).

As a side effect, the capture input capability of these lines can also be used in the address bus mode. Hereby changes of the upper address lines could be detected and trigger an interrupt request in order to perform some special service routines. External capture signals can only be applied if no address output is selected for PORT1.

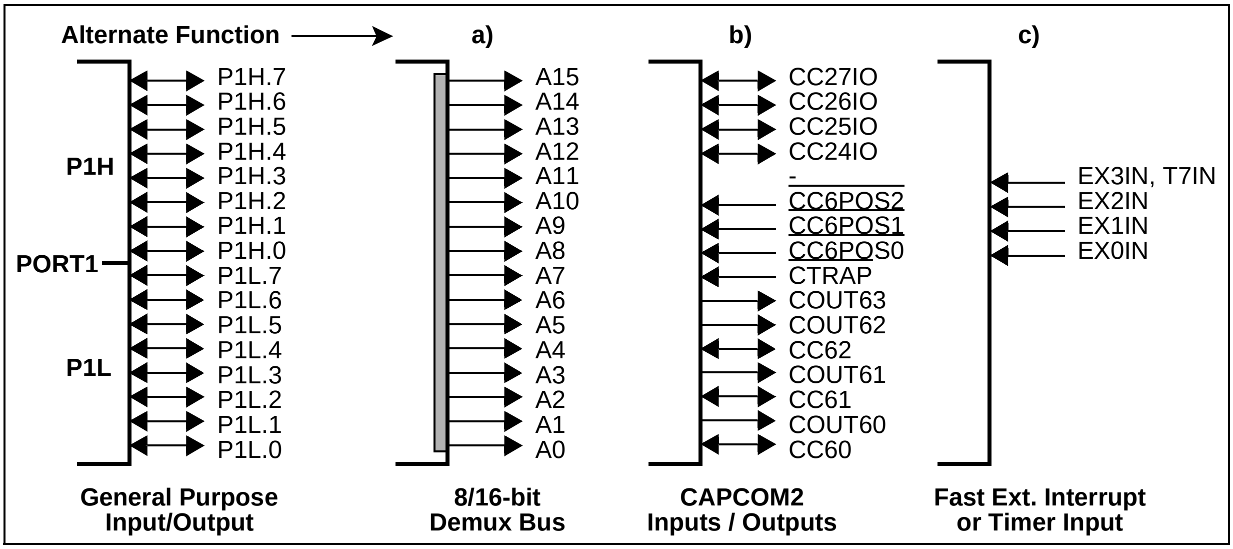


Figure 7-8 PORT1 IO and Alternate Functions

During external accesses in demultiplexed bus modes PORT1 outputs the 16-bit intra-segment address as an alternate output function.

During external accesses in multiplexed bus modes, when **no** BUSCON register selects a demultiplexed bus mode, PORT1 is not used and is available for general purpose IO.

When an external bus mode is enabled, the direction of the port pin and the loading of data into the port output latch are controlled by the bus controller hardware. The input of the port output latch is disconnected from the internal bus and is switched to the line labeled “Alternate Data Output” via a multiplexer. The alternate data is the 16-bit intrasegment address. While an external bus mode is enabled, the user software should not write to the port output latch, otherwise unpredictable results may occur. When the external bus modes are disabled, the contents of the direction register last written by the user becomes active.

The figures below show the structure of a PORT1 pins. The upper 4 pins of PORT1 combine internal bus data and alternate data output before the port latch input.

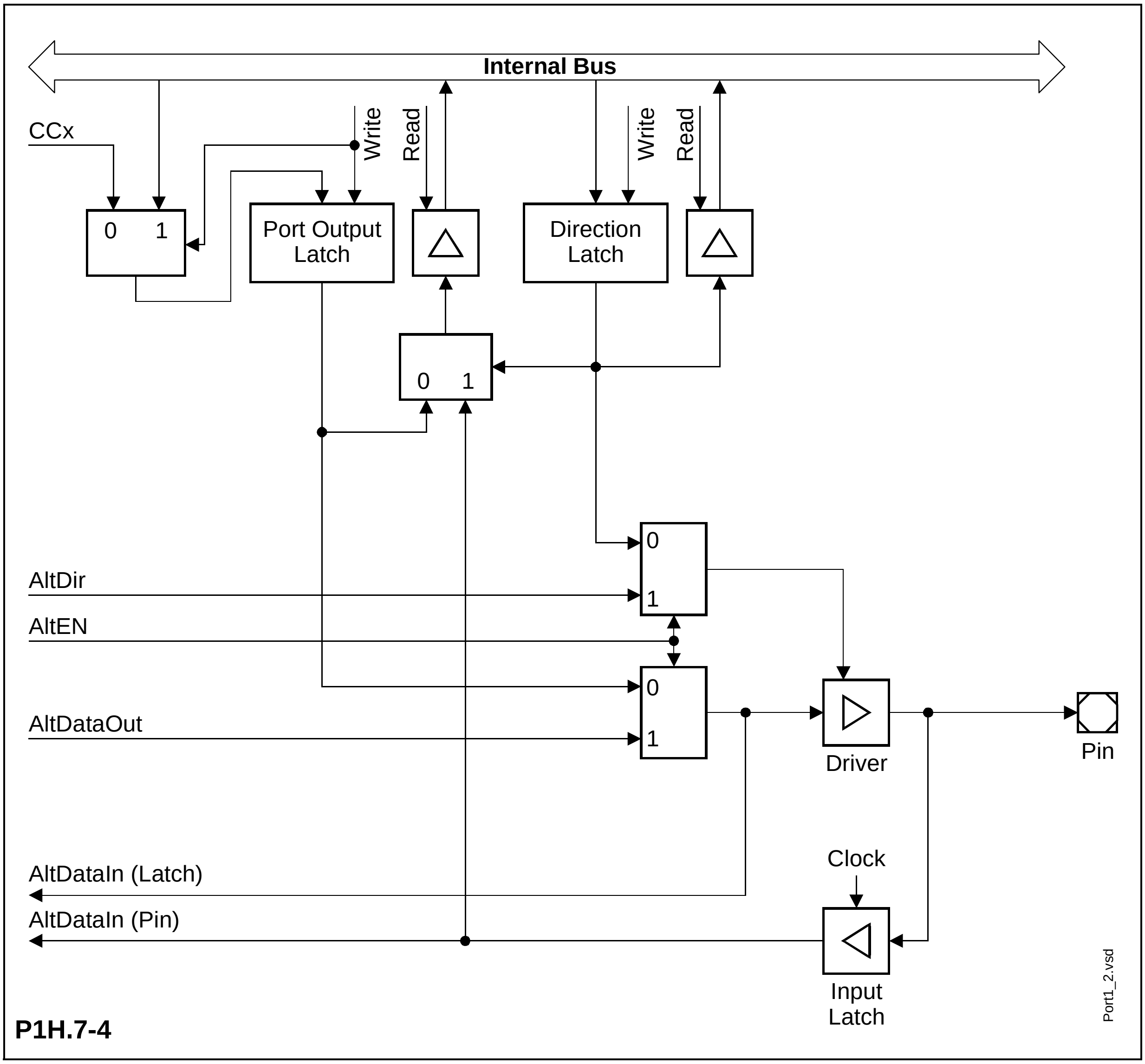


Figure 7-9 Block Diagram of a PORT1 Pin with Address and CAPCOM Function

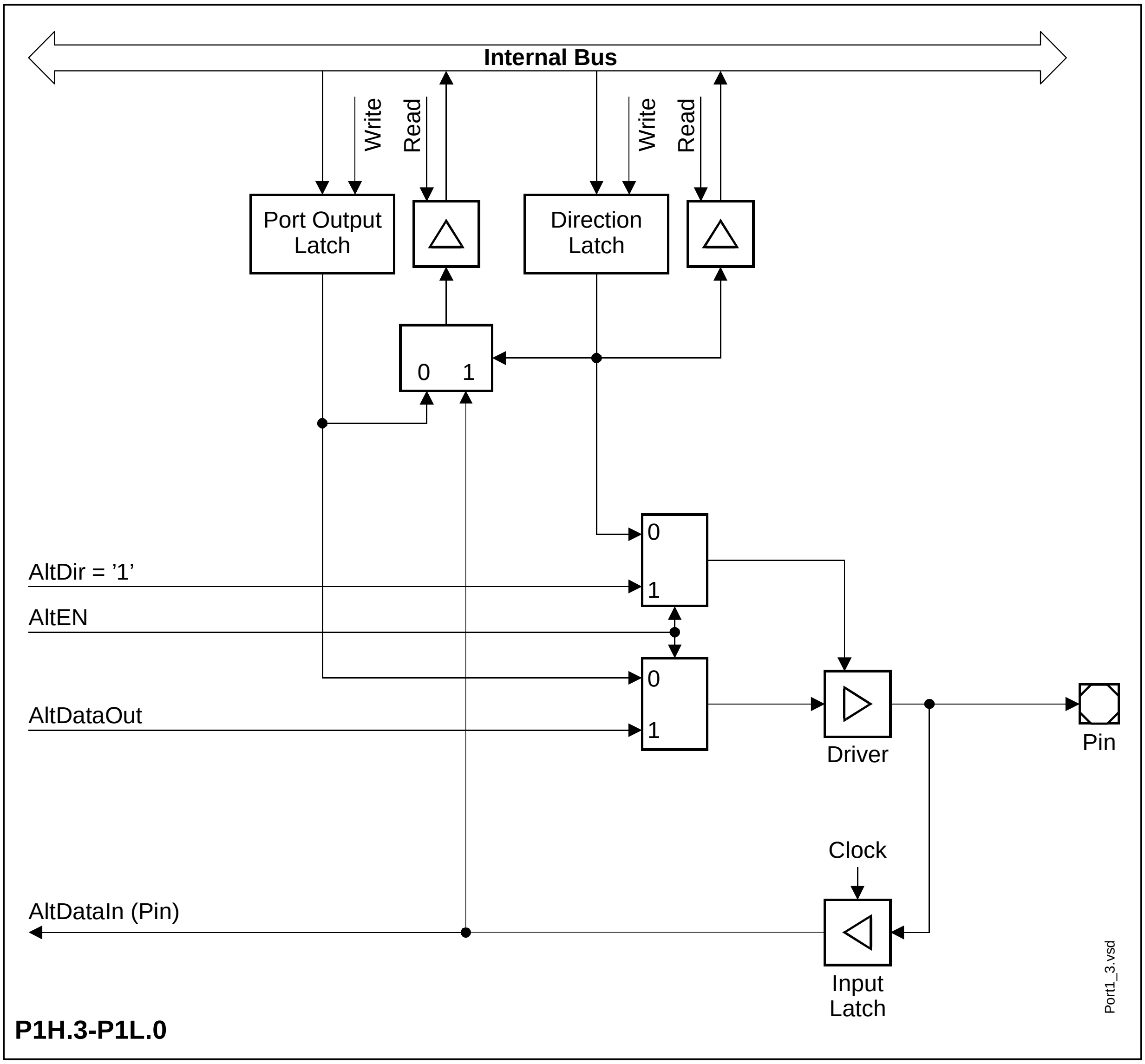


Figure 7-10 Block Diagram of a PORT1 Pin with Address and Alternate Input/Output Function

7.6 Port 3

If this 9-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP3. Each port line can be switched into push/pull or open drain mode via the open drain control register ODP3 (pins P3.15 and P3.12 do not support open drain mode!).

P3
Port 3 Data Register **SFR (FFC4$_H$/E2$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P3 .15	**-**	**P3 .13**	**P3 .12**	**P3 .11**	**P3 .10**	**P3 .9**	**P3 .8**	**-**	**P3.6**	**-**	**P3.4**	**-**	**-**	**-**	**-**
rw	-	rw	rw	rw	rw	rw	rw	-	rw	-	rw	-	-	-	-

Bit	Function
P3.y	**Port data register P3 bit y**

DP3
P3 Direction Ctrl. Register **SFR (FFC6$_H$/E3$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DP3 .15	**-**	**DP3 .13**	**DP3 .12**	**DP3 .11**	**DP3 .10**	**DP3 .9**	**DP3 .8**	**-**	**DP3 .6**	**-**	**DP3 .4**	**-**	**-**	**-**	**-**
rw	-	rw	rw	rw	rw	rw	rw	-	rw	-	rw	-	-	-	-

Bit	Function
DP3.y	**Port direction register DP3 bit y** DP3.y = 0: Port line P3.y is an input (high-impedance) DP3.y = 1: Port line P3.y is an output

ODP3
P3 Open Drain Ctrl. Reg. **ESFR (F1C6$_H$/E3$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**-**	**ODP 3.13**	**-**	**ODP 3.11**	**ODP 3.10**	**ODP 3.9**	**ODP 3.8**	**-**	**ODP 3.6**	**-**	**ODP 3.4**	**-**	**-**	**-**	**-**
-	-	rw	-	rw	rw	rw	rw	-	rw	-	rw	-	-	-	-

Bit	Function
ODP3.y	**Port 3 Open Drain control register bit y** ODP3.y = 0: Port line P3.y output driver in push/pull mode ODP3.y = 1: Port line P3.y output driver in open drain mode

*Note: Due to pin limitations register bit P3.14 is not connected to an IO pin.
Pins P3.15 and P3.12 do not support open drain mode.*

Alternate Functions of Port 3

The pins of Port 3 serve for various functions which include external timer control lines, the two serial interfaces, and the control lines BHE/WRH and CLKOUT/FOUT.

The table below summarizes the alternate functions of Port 3.

Table 7-4 Alternate Functions of Port 3

Port 3 Pin	Alternate Function	
-	-	
P3.4	T3EUD	Timer 3 External Up/Down Input
-	-	
P3.6	T3IN	Timer 3 Count Input
-	-	
P3.8	MRST	SSC Master Receive / Slave Transmit
P3.9	MTSR	SSC Master Transmit / Slave Receive
P3.10	TxD0	ASC0 Transmit Data Output
P3.11	RxD0	ASC0 Receive Data Input
P3.12	BHE/WRH	Byte High Enable / Write High Output
P3.13	SCLK	SSC Shift Clock Input/Output
-	-	
P3.15	CLKOUT/FOUT	System Clock Output / Progr. Frequency Output

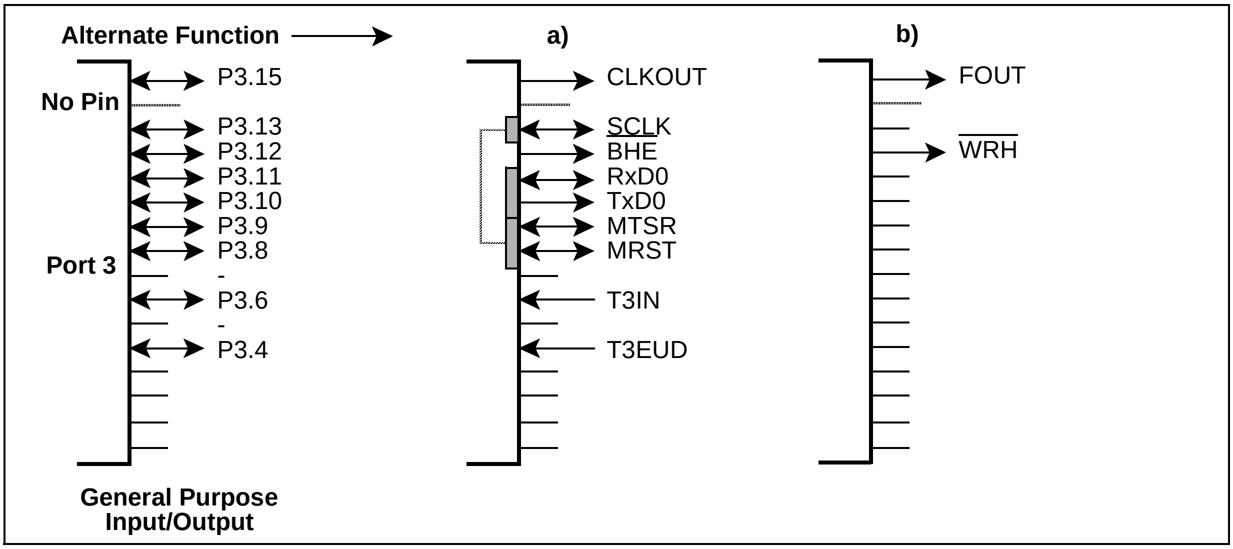


Figure 7-11 Port 3 IO and Alternate Functions

The port structure of the Port 3 pins depends on their alternate function (see fig. below).

When the on-chip peripheral associated with a Port 3 pin is configured to use the alternate input function, it reads the input latch, which represents the state of the pin, via the line labeled "Alternate Data Input". Port 3 pins with alternate input functions are:
T3IN and T3EUD.

When the on-chip peripheral associated with a Port 3 pin is configured to use the alternate output function, its "Alternate Data Output" line is ANDed with the port output latch line. When using these alternate functions, the user must set the direction of the port line to output (DP3.y=1) and must set the port output latch (P3.y=1). Otherwise the pin is in its high-impedance state (when configured as input) or the pin is stuck at '0' (when the port output latch is cleared). When the alternate output functions are not used, the "Alternate Data Output" line is in its inactive state, which is a high level ('1').
Port 3 pins with alternate output functions are:
TxD0 and CLKOUT/FOUT.

When the on-chip peripheral associated with a Port 3 pin is configured to use both the alternate input and output function, the descriptions above apply to the respective current operating mode. The direction must be set accordingly.
Port 3 pins with alternate input/output functions are:
MTSR, MRST, RxD0 and SCLK.

Note: Enabling the CLKOUT function automatically enables the P3.15 output driver. Setting bit DP3.15='1' is not required.
The CLKOUT function is automatically enabled in emulation mode.

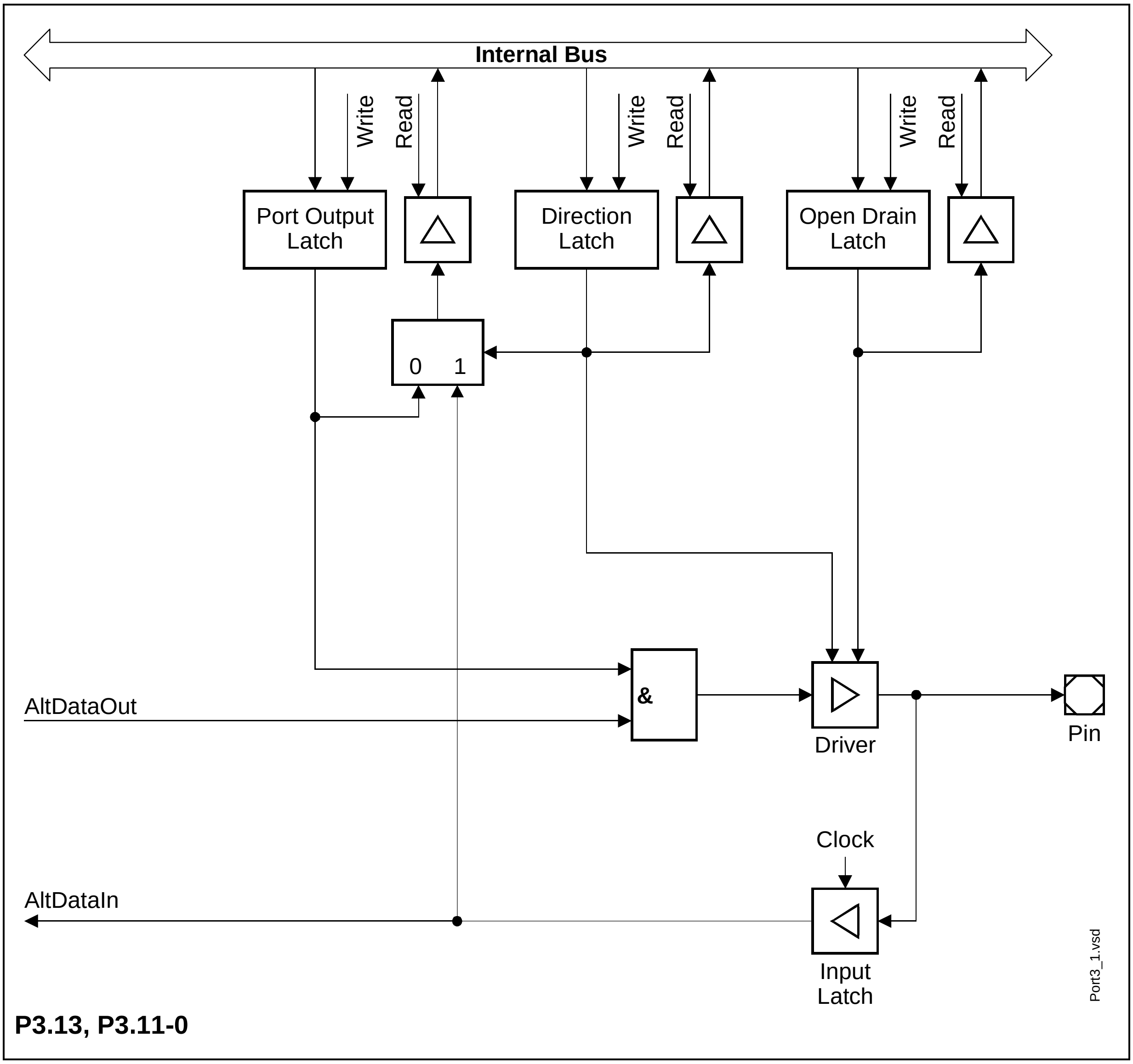


Figure 7-12 Block Diagram of a Port 3 Pin with Alternate Input or Alternate Output Function

Pin P3.12 (BHE/WRH) is one more pin with an alternate output function. However, its structure is slightly different (see figure below), because after reset the BHE or WRH function must be used depending on the system startup configuration. In these cases there is no possibility to program any port latches before. Thus the appropriate alternate function is selected automatically. If BHE/WRH is not used in the system, this pin can be used for general purpose IO by disabling the alternate function (BYTDIS = ‘1’ / WRCFG=’0’).

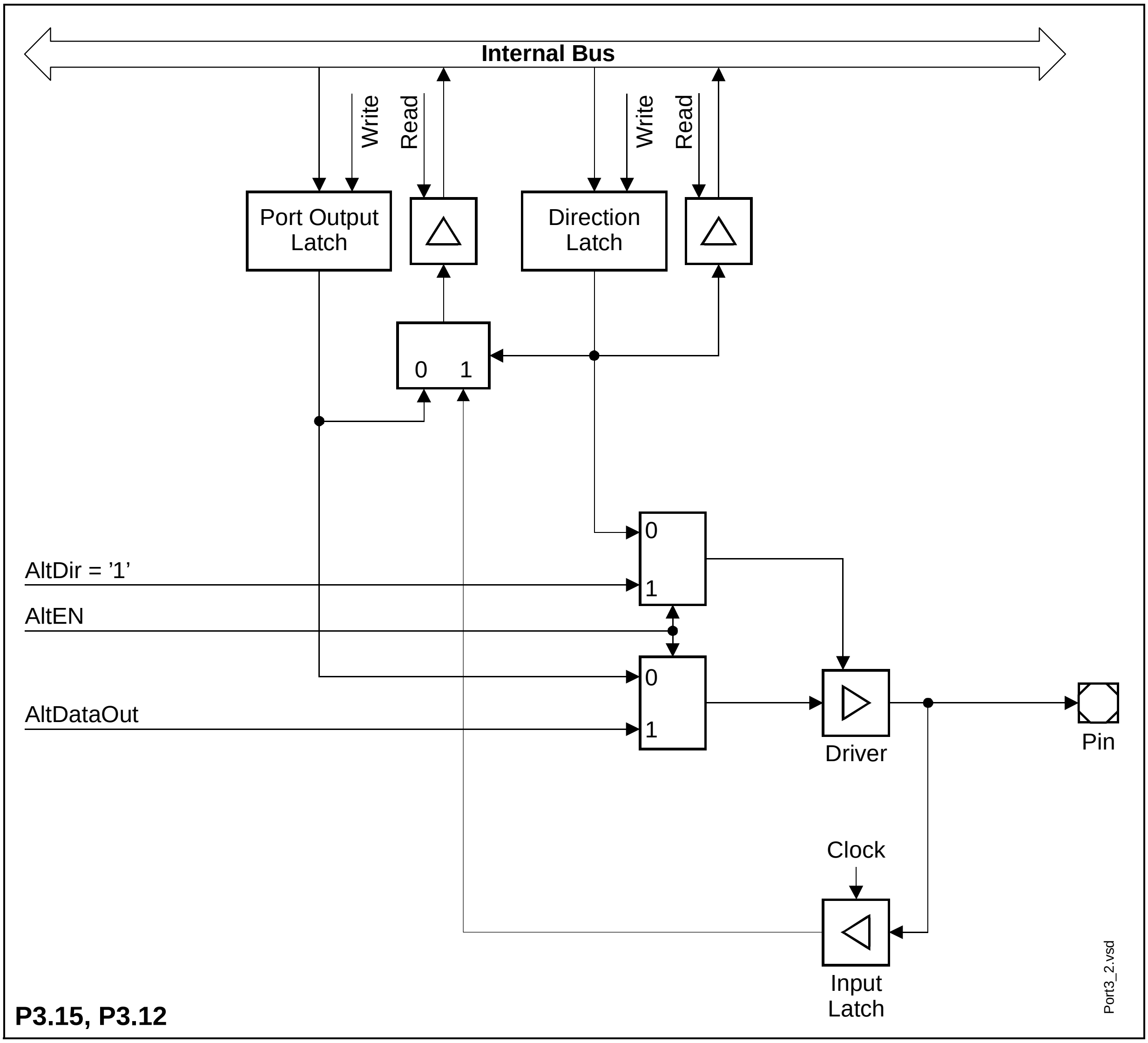


Figure 7-13 Block Diagram of Pins P3.15 (CLKOUT/FOUT) and P3.12 (BHE/WRH)

Note: Enabling the BHE or WRH function automatically enables the P3.12 output driver. Setting bit DP3.12=’1’ is not required.
Enabling the CLKOUT function automatically enables the P3.15 output driver. Setting bit DP3.15=’1’ is not required.

7.7 Port 4

If this 6-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP4.

P4
Port 4 Data Register **SFR (FFC8$_H$/E4$_H$)** **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	P4.6	P4.5	-	P4.3	P4.2	P4.1	P4.0
-	-	-	-	-	-	-	-	-	rw	rw	-	rw	rw	rw	rw

Bit	Function
P4.y	**Port data register P4 bit y**

DP4
P4 Direction Ctrl. Register **SFR (FFCA$_H$/E5$_H$)** **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	DP4 .6	DP4 .5	-	DP4 .3	DP4 .2	DP4 .1	DP4 .0
-	-	-	-	-	-	-	-	-	rw	rw	-	rw	rw	rw	rw

Bit	Function
DP4.y	**Port direction register DP4 bit y** DP4.y = 0: Port line P4.y is an input (high-impedance) DP4.y = 1: Port line P4.y is an output

ODP4
P4 Open Drain Ctrl. Reg. **ESFR (F1CA$_H$/E5$_H$)** **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	ODP 4.6	ODP 4.5	-	ODP 4.3	ODP 4.2	ODP 4.1	ODP 4.0
-	-	-	-	-	-	-	-	-	rw	rw	-	rw	rw	rw	rw

Bit	Function
ODP4.y	**Port 4 Open Drain control register bit y** ODP4.y = 0: Port line P4.y output driver in push/pull mode ODP4.y = 1: Port line P4.y output driver in open drain mode

7.7.1 Alternate Functions of Port 4

During external bus cycles that use segmentation (i.e. an address space above 64 KByte) a number of Port 4 pins may output the segment address lines. The number of pins that is used for segment address output determines the external address space which is directly accessible. The other pins of Port 4 (if any) may be used for general purpose IO or for the CAN interface.

If segment address lines are selected, the alternate function of Port 4 may be necessary to access e.g. external memory directly after reset. For this reason Port 4 will be switched to this alternate function automatically.

The number of segment address lines is selected via PORT0 during reset. The selected value can be read from bitfield SALSEL or CSSEL in register RP0H (read only) e.g. in order to check the configuration during run time. Software can adjust the number of selected segment address lines via register RSTCON.

The CAN interface can use 2 pins of Port 4 to interface the CAN module to an external CAN transceiver. In this case the number of possible segment address lines is reduced.

The table below summarizes the alternate functions of Port 4 depending on the number of selected segment address lines (coded via bitfield SALSEL).

Table 7-5 Alternate Functions of Port 4

Port 4 Pin	Std. Function SALSEL=01 64 KB	Altern. Function SALSEL=11 256KB	Altern. Function SALSEL=00 1 MB	Altern. Function SALSEL=10 4 MB
P4.0	Gen. p. IO or $\overline{\text{CS3}}$	Seg. Addr. A16	Seg. Addr. A16	Seg. Addr. A16
P4.1	Gen. p. IO or $\overline{\text{CS2}}$	Seg. Addr. A17	Seg. Addr. A17	Seg. Addr. A17
P4.2	Gen. p. IO or $\overline{\text{CS1}}$	Gen. p. IO or $\overline{\text{CS1}}$	Seg. Addr. A18	Seg. Addr. A18
P4.3	Gen. p. IO or $\overline{\text{CS0}}$	Gen. p. IO or $\overline{\text{CS0}}$	Seg. Addr. A19	Seg. Addr. A19
-	-	-	-	-
P4.5	Gen. p. IO or CAN	Gen. p. IO or CAN	Gen. p. IO or CAN	S.A. A20 or CAN
P4.6	Gen. p. IO or CAN	Gen. p. IO or CAN	Gen. p. IO or CAN	S.A. A21 or CAN
-	-	-	-	-

Note: Port 4 pins that are neither used for segment address output nor for chip select output nor for the CAN interface may be used for general purpose IO. The pins which are used for chip select output are defined via bitfield CSSEL (see register RP0H).
If more than one function is selected for a Port 4 pin, the segment address takes preference over the chip select lines, the CAN interface takes preference over the segment address lines.

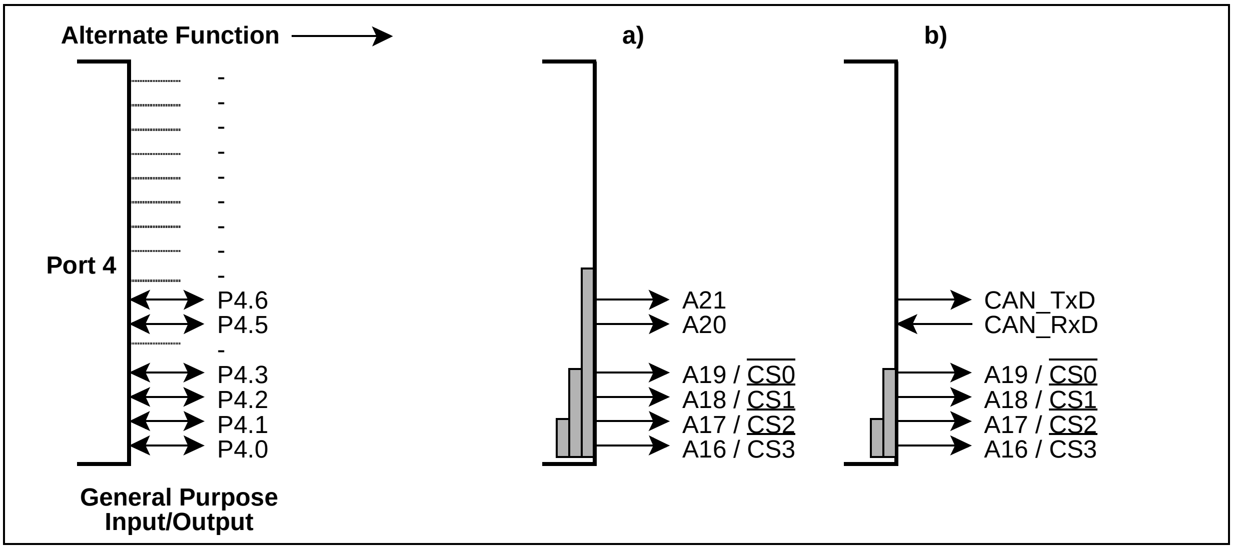

Alternate Function
a)
b)
Port 4
P4.6
P4.5
P4.3
P4.2
P4.1
P4.0
A21
A20
A19 / $\overline{\text{CS0}}$
A18 / $\overline{\text{CS1}}$
A17 / $\overline{\text{CS2}}$
A16 / CS3
CAN_TxD
CAN_RxD
A19 / $\overline{\text{CS0}}$
A18 / $\overline{\text{CS1}}$
A17 / $\overline{\text{CS2}}$
A16 / CS3
General Purpose
Input/Output

Figure 7-14 Port 4 IO and Alternate Functions

The chip select lines of Port 4 additionally have an internal weak pullup device which is switched on during any reset (including single-chip mode reset) in order to provide an inactive level on the optional chip select lines until the controller begins operation.

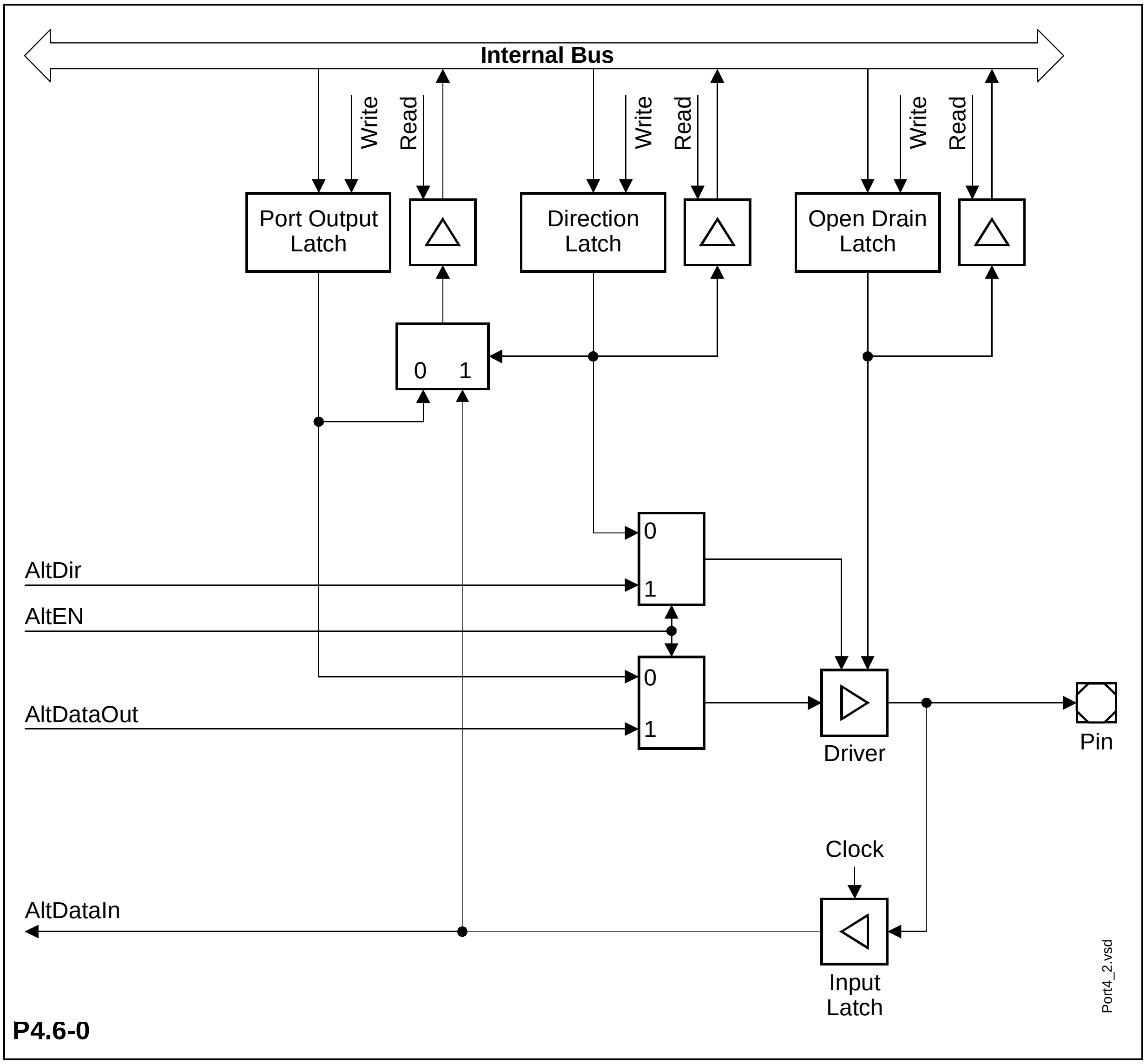


Figure 7-15 Block Diagram of a Port 4 Pin

7.8 Port 5

This 8-bit input port can only read data. There is no output latch and no direction register. Data written to P5 will be lost.

P5
Port 5 Data Register **SFR (FFA2$_H$/D1$_H$)** **Reset Value: XXXX$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**-**							**P5.7**	**P5.6**	**P5.5**	**P5.4**	**P5.3**	**P5.2**	**P5.1**	**P5.0**
-	-	-	-	-	-	-	-	r	r	r	r	r	r	r	r

Bit	Function
P5.y	**Port data register P5 bit y** (Read only)

Alternate Functions of Port 5

Each line of Port 5 is also connected to the input multiplexer of the Analog/Digital Converter. All port lines can accept analog signals (ANx) that can be converted by the ADC. For pins that shall be used as analog inputs it is recommended to disable the digital input stage via register P5DIDIS (see description below). This avoids undesired cross currents and switching noise while the (analog) input signal level is between V_{IL} and V_{IH}. Some pins of Port 5 also serve as external GPT timer control lines.

The table below summarizes the alternate functions of Port 5.

Table 7-6 Alternate Functions of Port 5

Port 5 Pin	Alternate Function a)	Alternate Function b)
P5.0	Analog Input AN0	-
P5.1	Analog Input AN1	-
P5.2	Analog Input AN2	-
P5.3	Analog Input AN3	-
P5.4	Analog Input AN4	T2EUD Timer 2 ext. Up/Down Input
P5.5	Analog Input AN5	T4EUD Timer 4 ext. Up/Down Input
P5.6	Analog Input AN6	T2IN Timer 2 Count Input
P5.7	Analog Input AN7	T4IN Timer 4 Count Input

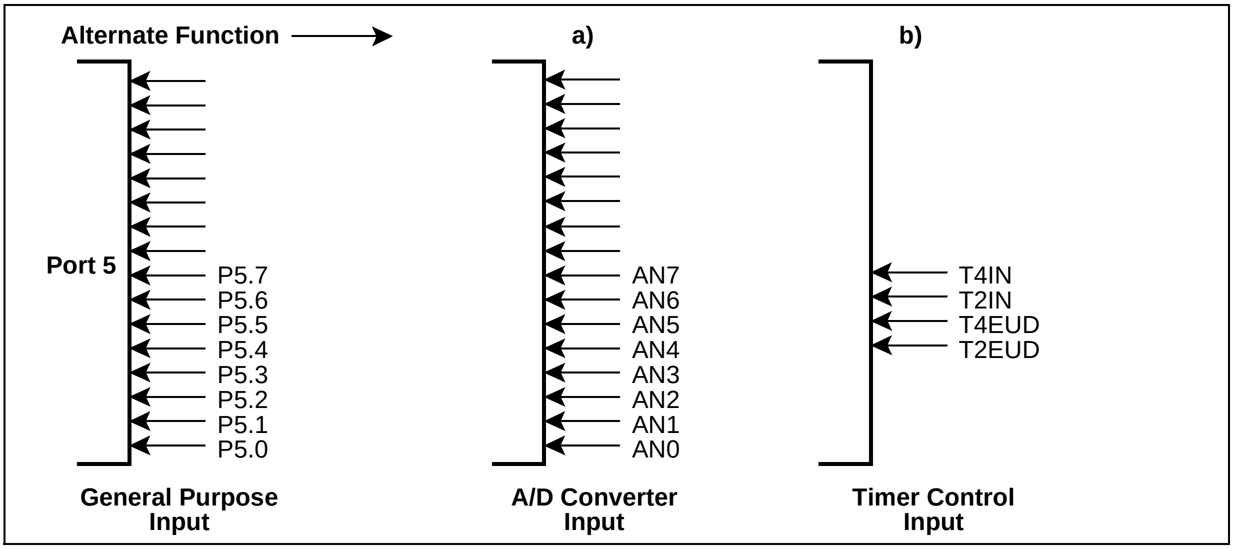


Figure 7-16 Port 5 IO and Alternate Functions

7.8.1 Port 5 Digital Input Control

Port 5 pins may be used for both digital an analog input. By setting the respective bit in register P5DIDIS the digital input stage of the respective Port 5 pin can be disconnected from the pin. This is recommended when the pin is to be used as analog input, as it reduces the current through the digital input stage and prevents it from toggling while the (analog) input level is between the digital low and high thresholds. So the consumed power and the generated noise can be reduced.

After reset all digital input stages are enabled.

P5DIDIS
P5 Dig. Inp. Disable Reg. SFR (FFA4$_H$/D2$_H$) Reset Value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	P5D .7	P5D .6	P5D .5	P5D .4	P5D .3	P5D .2	P5D .1	P5D .0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
P5D.y	**Port P5 Bit y Digital Input Control** 0: Digital input stage connected to port line P5.y 1: Digital input stage disconnected from port line P5.y *Note: When being read or used as alternate input this line appears as '1'.*

Port 5 pins have a special port structure (see figure below), first because it is an input only port, and second because the analog input channels are directly connected to the pins rather than to the input latches.

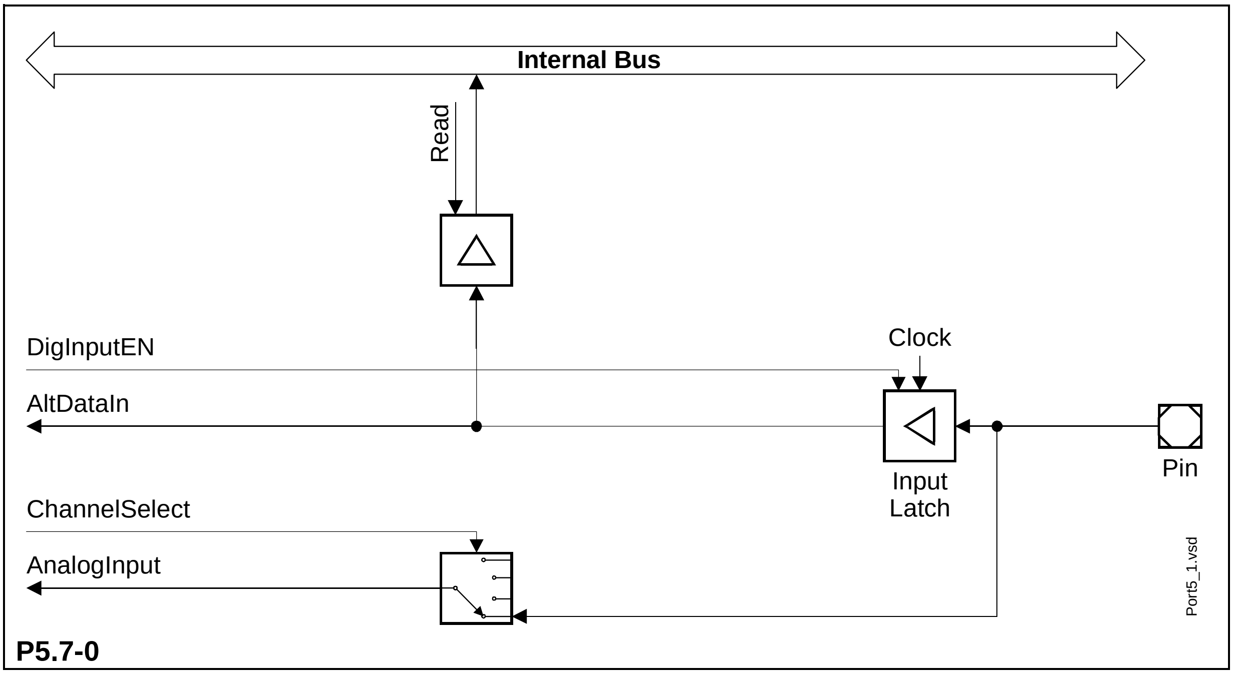


Figure 7-17 Block Diagram of a Port 5 Pin

Note: The line “AltDataIn” does not exist on all Port 5 inputs.

]

7.9 Port 8

If this 4-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP8. Each port line can be switched into push/pull or open drain mode via the open drain control register ODP8.

P8
Port 8 Data Register **SFR (FFD4$_H$/EA$_H$)** **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	-	-	-	**P8.3**	**P8.2**	**P8.1**	**P8.0**
-	-	-	-	-	-	-	-	-	-	-	-	rwh	rwh	rwh	rwh

Bit	Function
P8.y	**Port data register P8 bit y**

DP8
P8 Direction Ctrl. Register **SFR (FFD6$_H$/EB$_H$)** **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	-	-	-	**DP8 .3**	**DP8 .2**	**DP8 .1**	**DP8 .0**
-	-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw

Bit	Function
DP8.y	**Port direction register DP8 bit y** DP8.y = 0: Port line P8.y is an input (high-impedance) DP8.y = 1: Port line P8.y is an output

ODP8
P8 Open Drain Ctrl. Reg. **ESFR (F1D6$_H$/EB$_H$)** **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	-	-	-	**ODP8 .3**	**ODP8 .2**	**ODP8 .1**	**ODP8 .0**
-	-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw

Bit	Function
ODP8.y	**Port 8 Open Drain control register bit y** ODP8.y = 0: Port line P8.y output driver in push/pull mode ODP8.y = 1: Port line P8.y output driver in open drain mode

7.9.1 Alternate Functions of Port 8

All Port 8 lines serve as capture inputs or compare outputs (CCxIO) for the CAPCOM2 unit (see table below).

When a Port 8 line is used as a capture input, the state of the input latch, which represents the state of the port pin, is directed to the CAPCOM unit via the line “Alternate Pin Data Input”. If an external capture trigger signal is used, the direction of the respective pin must be set to input. If the direction is set to output, the state of the port output latch will be read since the pin represents the state of the output latch. This can be used to trigger a capture event through software by setting or clearing the port latch. Note that in the output configuration, no external device may drive the pin, otherwise conflicts would occur.

When a Port 8 line is used as a compare output (compare modes 1 and 3), the compare event (or the timer overflow in compare mode 3) directly effects the port output latch. In compare mode 1, when a valid compare match occurs, the state of the port output latch is read by the CAPCOM control hardware via the line “Alternate Latch Data Input”, inverted, and written back to the latch via the line “Alternate Data Output”. The port output latch is clocked by the signal “Compare Trigger” which is generated by the CAPCOM unit. In compare mode 3, when a match occurs, the value ’1’ is written to the port output latch via the line “Alternate Data Output”. When an overflow of the corresponding timer occurs, a ’0’ is written to the port output latch. In both cases, the output latch is clocked by the signal “Compare Trigger”. The direction of the pin should be set to output by the user, otherwise the pin will be in the high-impedance state and will not reflect the state of the output latch.

As can be seen from the port structure below, the user software always has free access to the port pin even when it is used as a compare output. This is useful for setting up the initial level of the pin when using compare mode 1 or the double-register mode. In these modes, unlike in compare mode 3, the pin is not set to a specific value when a compare match occurs, but is toggled instead.

When the user wants to write to the port pin at the same time a compare trigger tries to clock the output latch, the write operation of the user software has priority. Each time a CPU write access to the port output latch occurs, the input multiplexer of the port output latch is switched to the line connected to the internal bus. The port output latch will receive the value from the internal bus and the hardware triggered change will be lost.

As all other capture inputs, the capture input function of the Port 8 pins can also be used as external interrupt inputs (sample rate 16 TCL).

The CAN interface can use 2 pins of Port 8 to interface the CAN Module to an external transceiver. In this case the number of possible CAPCOM IO lines is reduced.

Table 7-7 Alternate Functions of Port 8

Port 8 Pin	Alternate Function	
P8.0	CC16IO	Capture input / compare output channel 16 or CAN
P8.1	CC17IO	Capture input / compare output channel 17 or CAN
P8.2	CC18IO	Capture input / compare output channel 18 or CAN
P8.3	CC19IO	Capture input / compare output channel 19 or CAN

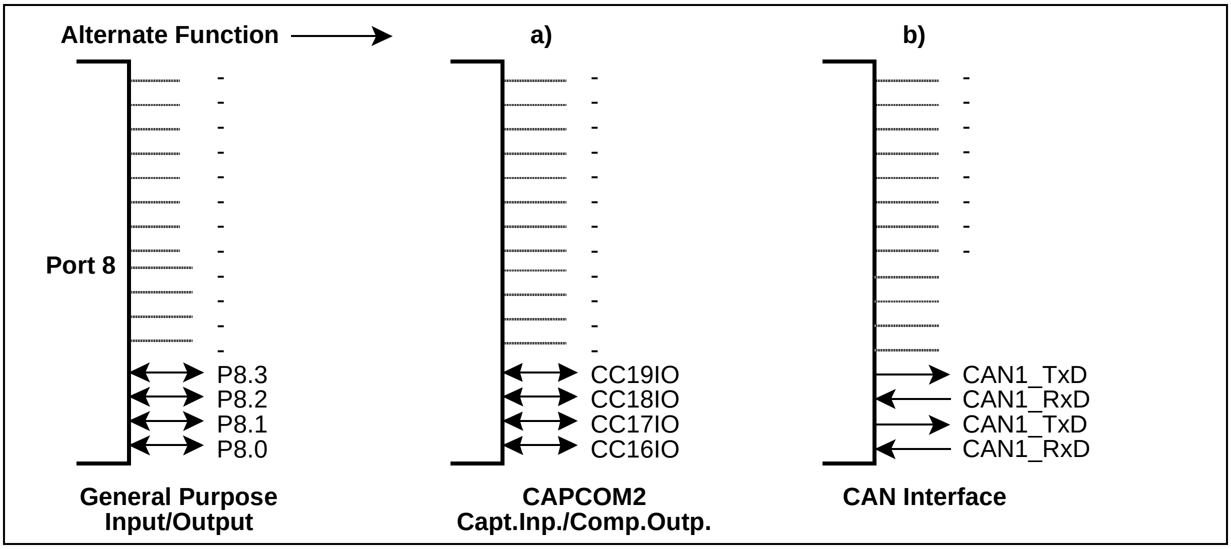


Figure 7-18 Port 8 IO and Alternate Functions

Note: The usage of Port 8 pins for CAN interface lines depends on the chosen assignment for the CAN module.
CAN interface lines will override general purpose IO and CAPCOM IO lines.

The pins of Port 8 combine internal bus data and alternate data output before the port latch input.

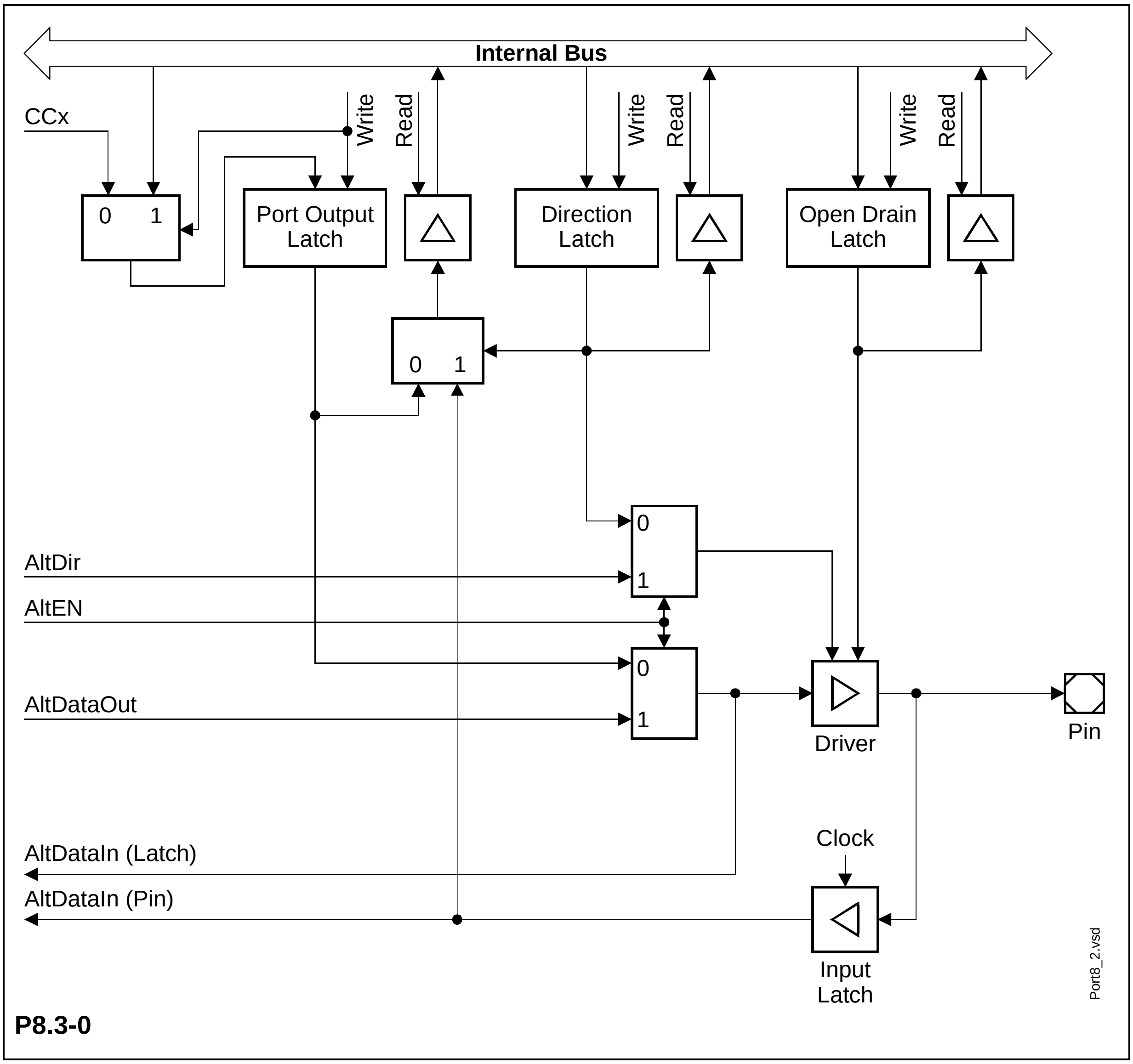


Figure 7-19 Block Diagram of Port 8 Pins with an Alternate CAPCOM IO and CAN Interface Function

8 Dedicated Pins

Most of the input/output or control signals of the functional the C164 are realized as alternate functions of pins of the parallel ports. There is, however, a number of signals that use separate pins, including the oscillator, special control signals and, of course, the power supply.

The table below summarizes the 21 dedicated pins of the C164.

Table 8-1 C164 Dedicated Pins

Pin(s)	Function
ALE	Address Latch Enable
$\overline{\mathrm{RD}}$	External Read Strobe
$\overline{\mathrm{WR}}/\overline{\mathrm{WRL}}$	External Write/Write Low Strobe
$\overline{\mathrm{EA}}$/ VPP	External Access Enable and External Programming Voltage
$\overline{\mathrm{NMI}}$	Non-Maskable Interrupt Input
XTAL1, XTAL2	Oscillator Input / Output
$\overline{\mathrm{RSTIN}}$	Reset Input
$\overline{\mathrm{RSTOUT}}$	Reset Output
VAREF, VAGND	Power Supply for Analog/Digital Converter
VDD, VSS	Digital Power Supply and Ground (5 pins each)

The Address Latch Enable signal ALE controls external address latches that provide a stable address in multiplexed bus modes.
ALE is activated for every external bus cycle independent of the selected bus mode, i.e. it is also activated for bus cycles with a demultiplexed address bus. When an external bus is enabled (one or more of the BUSACT bits set) also X-Peripheral accesses will generate an active ALE signal.
ALE is not activated for internal accesses, i.e. accesses to ROM/OTP/Flash (if provided), the internal RAM and the special function registers. In single chip mode, i.e. when no external bus is enabled (no BUSACT bit set), ALE will also remain inactive for X-Peripheral accesses.
During reset an internal pulldown ensures an inactive (low) level on the ALE output.

At the end of reset the current level on pin ALE is latched and is used for configuration (together with pin $\overline{\mathrm{RD}}$). Pin ALE selects standard start/boot, when driven low (default) or alternate start/boot when driven high.

For standard configuration pin ALE should be low or not connected.

The External Read Strobe $\overline{\text{RD}}$ controls the output drivers of external memory or peripherals when the C164 reads data from these external devices. During accesses to on-chip X-Peripherals $\overline{\text{RD}}$ remains inactive (high).
During reset an internal pullup ensures an inactive (high) level on the $\overline{\text{RD}}$ output.

At the end of reset the current level on pin $\overline{\text{RD}}$ is latched and is used for configuration.

For a reset with external access ($\overline{\text{EA}}$=’0’) pin $\overline{\text{RD}}$ controls the oscillator watchdog. The latched $\overline{\text{RD}}$ level determines the reset value of bit OWDDIS in register SYSCON. The default high level on pin $\overline{\text{RD}}$ leaves the oscillator watchdog active (OWDDIS=’0’), while a low level disables the watchdog (OWDDIS=’1’) e.g. for testing purposes.

For a true single-chip mode reset ($\overline{\text{EA}}$=’1’) pin $\overline{\text{RD}}$ enables the bootstrap loader, when driven low (pin ALE is evaluated together with pin $\overline{\text{RD}}$).

For standard configuration pin $\overline{\text{RD}}$ should be high or not connected.

The External Write Strobe $\overline{\text{WR}}$/$\overline{\text{WRL}}$ controls the data transfer from the C164 to an external memory or peripheral device. This pin may either provide an general $\overline{\text{WR}}$ signal activated for both byte and word write accesses, or specifically control the low byte of an external 16-bit device ($\overline{\text{WRL}}$) together with the signal $\overline{\text{WRH}}$ (alternate function of P3.12/$\overline{\text{BHE}}$). During accesses to on-chip X-Peripherals $\overline{\text{WR}}$/$\overline{\text{WRL}}$ remains inactive (high).
During reset an internal pullup ensures an inactive (high) level on the $\overline{\text{WR}}$/$\overline{\text{WRL}}$ output.

The External Access Enable Pin $\overline{\text{EA}}$ determines if the C164 after reset starts fetching code from the internal ROM area ($\overline{\text{EA}}$=’1’) or via the external bus interface ($\overline{\text{EA}}$=’0’). Be sure to hold this input low for ROMless devices. At the end of the internal reset sequence the $\overline{\text{EA}}$ signal is latched together with the configuration (PORT0, $\overline{\text{RD}}$, ALE).

The Non-Maskable Interrupt Input $\overline{\text{NMI}}$ allows to trigger a high priority trap via an external signal (e.g. a power-fail signal). It also serves to validate the PWRDN instruction that switches the C164 into Power-Down mode. The $\overline{\text{NMI}}$ pin is sampled with every CPU clock cycle to detect transitions.

The Oscillator Input XTAL1 and Output XTAL2 connect the internal Pierce oscillator to the external crystal. The oscillator provides an inverter and a feedback element. The standard external oscillator circuitry (see chapter „Clock Generation“) comprises the crystal, two low end capacitors and series resistor to limit the current through the crystal.

An external clock signal may be fed to the input XTAL1, leaving XTAL2 open or terminating it for higher input frequencies.

The Reset Input $\overline{\textbf{RSTIN}}$ allows to put the C164 into the well defined reset condition either at power-up or external events like a hardware failure or manual reset. The input voltage threshold of the $\overline{\text{RSTIN}}$ pin is raised compared to the standard pins in order to minimize the noise sensitivity of the reset input.

In bidirectional reset mode the C164's line $\overline{\text{RSTIN}}$ may be be driven active by the chip logic e.g. in order to support external equipment which is required for startup (e.g. flash memory).

Bidirectional reset reflects internal reset sources (software, watchdog) also to the $\overline{\text{RSTIN}}$ pin and converts short hardware reset pulses to a minimum duration of the internal reset sequence. Bidirectional reset is enabled by setting bit BDRSTEN in register SYSCON and changes $\overline{\text{RSTIN}}$ from a pure input to an open drain IO line. When an internal reset is triggered by the SRST instruction or by a watchdog timer overflow or a low level is applied to the $\overline{\text{RSTIN}}$ line, an internal driver pulls it low for the duration of the internal reset sequence. After that it is released and is then controlled by the external circuitry alone.

The bidirectional reset function is useful in applications where external devices require a defined reset signal but cannot be connected to the C164's $\overline{\text{RSTOUT}}$ signal, e.g. an external flash memory which must come out of reset and deliver code well before $\overline{\text{RSTOUT}}$ can be deactivated via EINIT.

The following behaviour differences must be observed when using the bidirectional reset feature in an application:

- Bit BDRSTEN in register SYSCON cannot be changed after EINIT and is cleared automatically after a reset.
- The reset indication flags always indicate a long hardware reset.
- The PORT0 configuration is treated like on a hardware reset. Especially the bootstrap loader may be activated when P0L.4 is low.
- Pin $\overline{\text{RSTIN}}$ may only be connected to external reset devices with an open drain output driver.
- A short hardware reset is extended to the duration of the internal reset sequence.

The Reset Output $\overline{\textbf{RSTOUT}}$ provides a special reset signal for external circuitry. $\overline{\text{RSTOUT}}$ is activated at the beginning of the reset sequence, triggered via $\overline{\text{RSTIN}}$, a watchdog timer overflow or by the SRST instruction. $\overline{\text{RSTOUT}}$ remains active (low) until the EINIT instruction is executed. This allows to initialize the controller before the external circuitry is activated.

Note: During emulation mode pin $\overline{\text{RSTOUT}}$ is used as an input and therefore must be driven by the external circuitry.

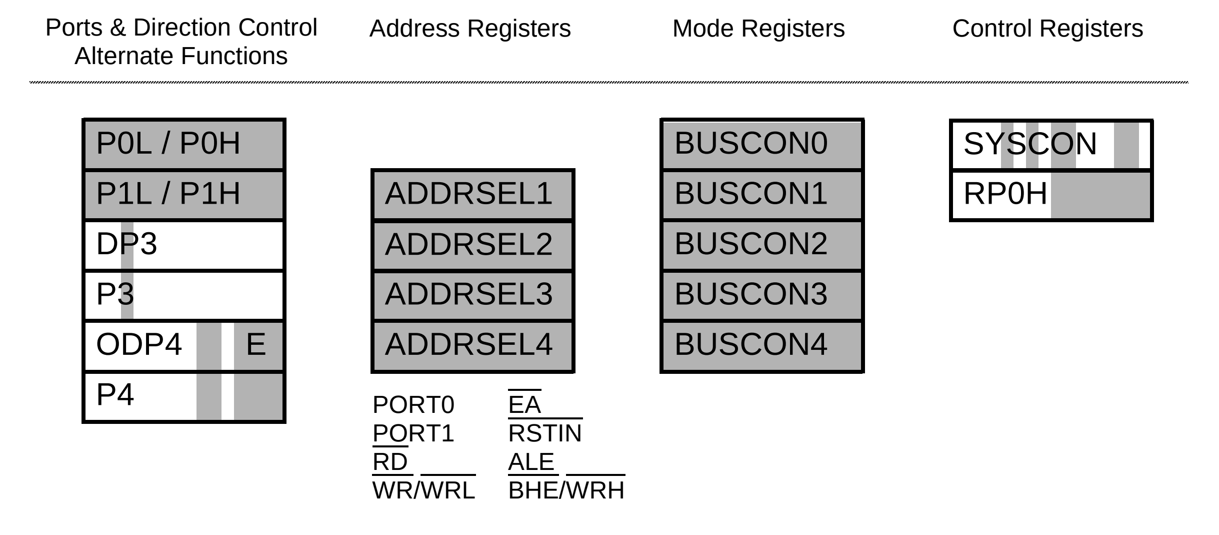
The Power Supply pins for the Analog/Digital Converter VAREF and VAGND provide a separate power supply for the on-chip ADC. This reduces the noise that is coupled to the analog input signals from the digital logic sections and so improves the stability of the conversion results, when VAREF and VAGND are properly discoupled from VDD and VSS.

The Power Supply pins VDD and VSS provide the power supply for the digital logic of the C164. The respective VDD/VSS pairs should be decoupled as close to the pins as possible. For best results it is recommended to implement two-level decoupling, e.g. (the widely used) 100 nF in parallel with 30...40 pF capacitors which deliver the peak currents.

Note: All VDD pins and all VSS pins must be connected to the power supply and ground, respectively.

9 The External Bus Interface

Although the C164 provides a powerful set of on-chip peripherals and on-chip RAM and ROM/OTP/Flash (except for ROMless versions) areas, these internal units only cover a small fraction of its address space of up to 16 MByte. The external bus interface allows to access external peripherals and additional volatile and non-volatile memory. The external bus interface provides a number of configurations, so it can be taylored to fit perfectly into a given application system.



P0L/P0H	PORT0 Data Registers	ADDRSELx	Address Range Select Register 1...4
P1L/P1H	PORT1 Data Registers	BUSCONx	Bus Mode Control Register 0...4
DP3	Port 3 Direction Control Register	SYSCON	System Control Register
P3	Port 3 Data Register	RP0H	Port P0H Reset Configuration Register
ODP4	Port 4 Open Drain Control Register		
P4	Port 4 Data Register		

Figure 9-1 SFRs and Port Pins Associated with the External Bus Interface

Accesses to external memory or peripherals are executed by the integrated External Bus Controller (EBC). The function of the EBC is controlled via the SYSCON register and the BUSCONx and ADDRSELx registers. The BUSCONx registers specify the external bus cycles in terms of data width (16-bit/8-bit), chip selects and length (waitstates / ALE / RW delay). These parameters are used for accesses within a specific address area which is defined via the corresponding register ADDRSELx.

The four pairs BUSCON1/ADDRSEL1...BUSCON4/ADDRSEL4 allow to define four independent “address windows”, while all external accesses outside these windows are controlled via register BUSCON0.

9.1 Single Chip Mode

Single chip mode is entered, when pin $\overline{\text{EA}}$ is high during reset (see also section "System Startup Configuration upon a Single-Chip Mode Reset"). In this case register BUSCON0 is initialized with 0000_H, which also resets bit BUSACT0, so no external bus is enabled.

In single chip mode the C164 operates only with and out of internal resources. No external bus is configured and no external peripherals and/or memory can be accessed. Also no port lines are occupied for the bus interface. When running in single chip mode, however, external access may be enabled by configuring an external bus under software control. Single chip mode allows the C164 to start execution out of the internal program memory (Mask-ROM, OTP or Flash memory).

Note: Any attempt to access a location in the external memory space in single chip mode results in the hardware trap ILLBUS if no external bus has been explicitly enabled by software.

9.2 External Bus Modes

When the external bus interface is enabled (bit BUSACTx='1') and configured (bitfield BTYP), the C164 uses a subset of its port lines together with some control lines to build the external bus.

Table 9-1 Summary of External Bus Modes

BTYP Encoding	External Data Bus Width	External Address Bus Mode
0 0	8-bit Data	Demultiplexed Addresses
0 1	8-bit Data	Multiplexed Addresses
1 0	16-bit Data	Demultiplexed Addresses
1 1	16-bit Data	Multiplexed Addresses

The bus configuration (BTYP) for the address windows (BUSCON4...BUSCON1) is selected via software typically during the initialization of the system.

The bus configuration (BTYP) for the default address range (BUSCON0) is selected via PORT0 during reset, provided that pin $\overline{\text{EA}}$ is low during reset. Otherwise BUSCON0 may be programmed via software just like the other BUSCON registers.

The 16 MByte address space of the C164 is divided into 256 segments of 64 KByte each. The 16-bit intra-segment address is output on PORT0. When segmentation is disabled, only one 64 KByte segment can be used and accessed. Otherwise additional address lines may be output on Port 4 (addressing up to 4 MByte) and/or several chip select lines may be used to select different memory banks or peripherals. These functions are selected during reset via bitfields SALSEL and CSSEL of register RP0H, respectively.

Note: Bit SGTDIS of register SYSCON defines, if the CSP register is saved during interrupt entry (segmentation active) or not (segmentation disabled).

Multiplexed Bus Modes

In the multiplexed bus modes the 16-bit intra-segment address as well as the data use PORT0. The address is time-multiplexed with the data and has to be latched externally. The width of the required latch depends on the selected data bus width, i.e. an 8-bit data bus requires a byte latch (the address bits A15...A8 on P0H do not change, while P0L multiplexes address and data), a 16-bit data bus requires a word latch (the least significant address line A0 is not relevant for word accesses).
The upper address lines (An...A16) are permanently output on Port 4 (if segmentation is enabled) and do not require latches.

The EBC initiates an external access by generating the Address Latch Enable signal (ALE) and then placing an address on the bus. The falling edge of ALE triggers an external latch to capture the address. After a period of time during which the address must have been latched externally, the address is removed from the bus. The EBC now activates the respective command signal ($\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{WRL}}$, $\overline{\text{WRH}}$). Data is driven onto the bus either by the EBC (for write cycles) or by the external memory/peripheral (for read cycles). After a period of time, which is determined by the access time of the memory/ peripheral, data become valid.

Read cycles: Input data is latched and the command signal is now deactivated. This causes the accessed device to remove its data from the bus which is then tri-stated again.

Write cycles: The command signal is now deactivated. The data remain valid on the bus until the next external bus cycle is started.

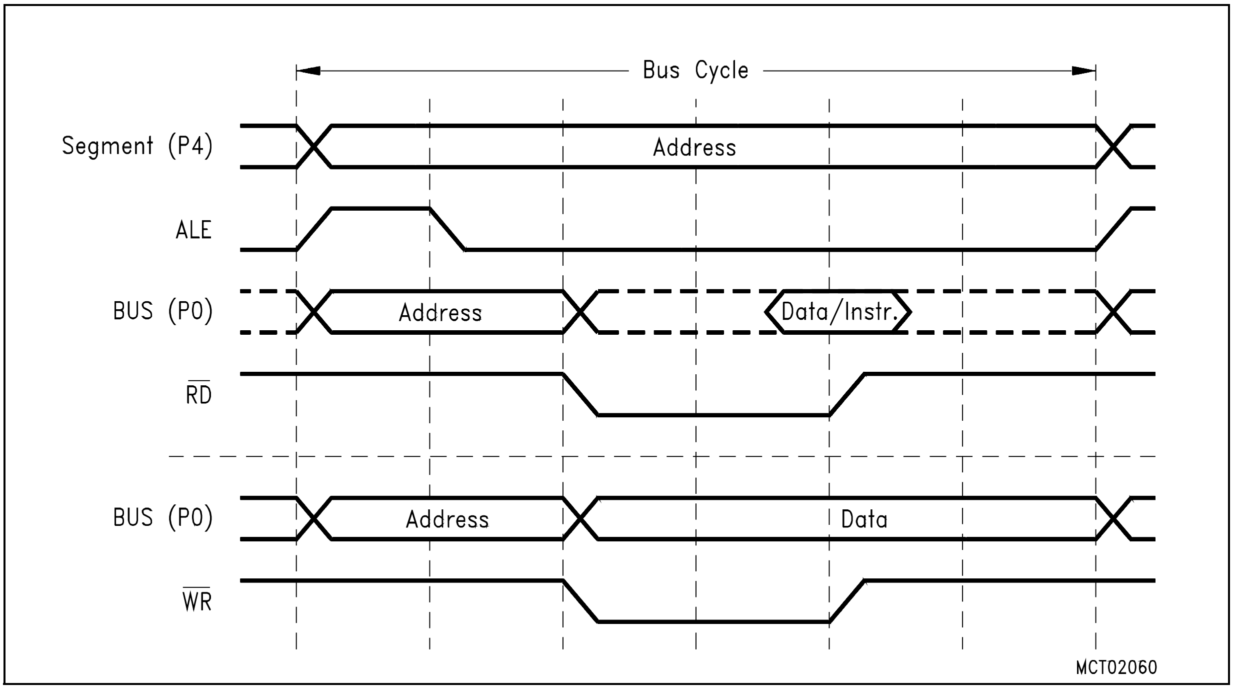


Figure 9-2 Multiplexed Bus Cycle

Demultiplexed Bus Modes

In the demultiplexed bus modes the 16-bit intra-segment address is permanently output on PORT1, while the data uses PORT0 (16-bit data) or P0L (8-bit data).
The upper address lines are permanently output on Port 4 (if selected via SALSEL during reset). No address latches are required.

The EBC initiates an external access by placing an address on the address bus. After a programmable period of time the EBC activates the respective command signal ($\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{WRL}}$, $\overline{\text{WRH}}$). Data is driven onto the data bus either by the EBC (for write cycles) or by the external memory/peripheral (for read cycles). After a period of time, which is determined by the access time of the memory/peripheral, data become valid.

Read cycles: Input data is latched and the command signal is now deactivated. This causes the accessed device to remove its data from the data bus which is then tri-stated again.

Write cycles: The command signal is now deactivated. If a subsequent external bus cycle is required, the EBC places the respective address on the address bus. The data remain valid on the bus until the next external bus cycle is started.

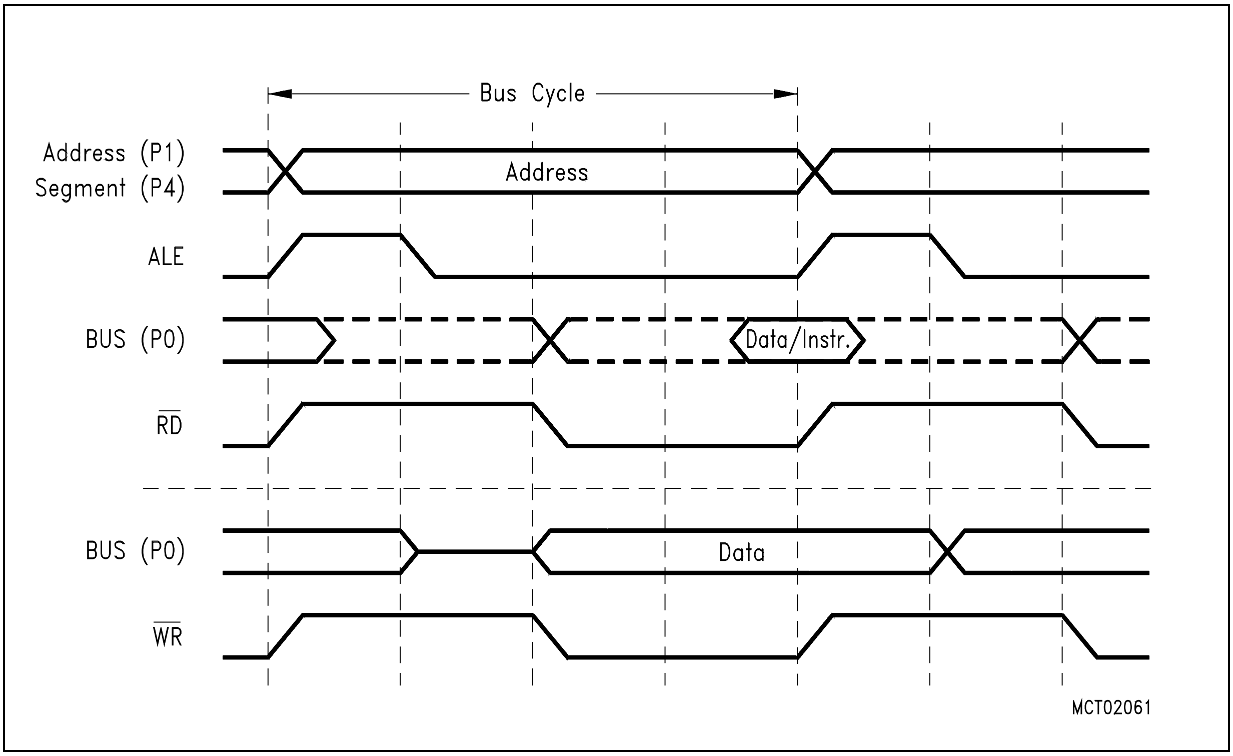


Figure 9-3 Demultiplexed Bus Cycle

Switching Between the Bus Modes

The EBC allows to switch between different bus modes dynamically, i.e. subsequent external bus cycles may be executed in different ways. Certain address areas may use an 8-bit or 16-bit data bus, or predefined waitstates.

A change of the external bus characteristics can be initiated in two different ways:

Reprogramming the BUSCON and/or ADDRSEL registers allows to either change the bus mode for a given address window, or change the size of an address window that uses a certain bus mode. Reprogramming allows to use a great number of different address windows (more than BUSCONs are available) on the expense of the overhead for changing the registers and keeping appropriate tables.

Switching between predefined address windows automatically selects the bus mode that is associated with the respective window. Predefined address windows allow to use different bus modes without any overhead, but restrict their number to the number of BUSCONs. However, as BUSCON0 controls all address areas, which are not covered by the other BUSCONs, this allows to have gaps between these windows, which use the bus mode of BUSCON0.

PORT1 will output the intra-segment address, when any of the BUSCON registers selects a demultiplexed bus mode, even if the current bus cycle uses a multiplexed bus mode. This allows to have an external address decoder connected to PORT1 only, while using it for all kinds of bus cycles.

Note: Never change the configuration for an address area that currently supplies the instruction stream. Due to the internal pipelining it is very difficult to determine the first instruction fetch that will use the new configuration. Only change the configuration for address areas that are not currently accessed. This applies to BUSCON registers as well as to ADDRSEL registers.

The usage of the BUSCON/ADDRSEL registers is controlled via the issued addresses. When an access (code fetch or data) is initiated, the respective generated physical address defines, if the access is made internally, uses one of the address windows defined by ADDRSEL4...1, or uses the default configuration in BUSCON0. After initializing the active registers, they are selected and evaluated automatically by interpreting the physical address. No additional switching or selecting is necessary during run time, except when more than the four address windows plus the default are to be used.

Switching from demultiplexed to multiplexed bus mode represents a special case. The bus cycle is started by activating ALE and driving the address to Port 4 and PORT1 as usual, if another BUSCON register selects a demultiplexed bus. However, in the multiplexed bus modes the address is also required on PORT0. In this special case the address on PORT0 is delayed by one CPU clock cycle, which delays the complete (multiplexed) bus cycle and extends the corresponding ALE signal (see figure below).

This extra time is required to allow the previously selected device (via demultiplexed bus) to release the data bus, which would be available in a demultiplexed bus cycle.

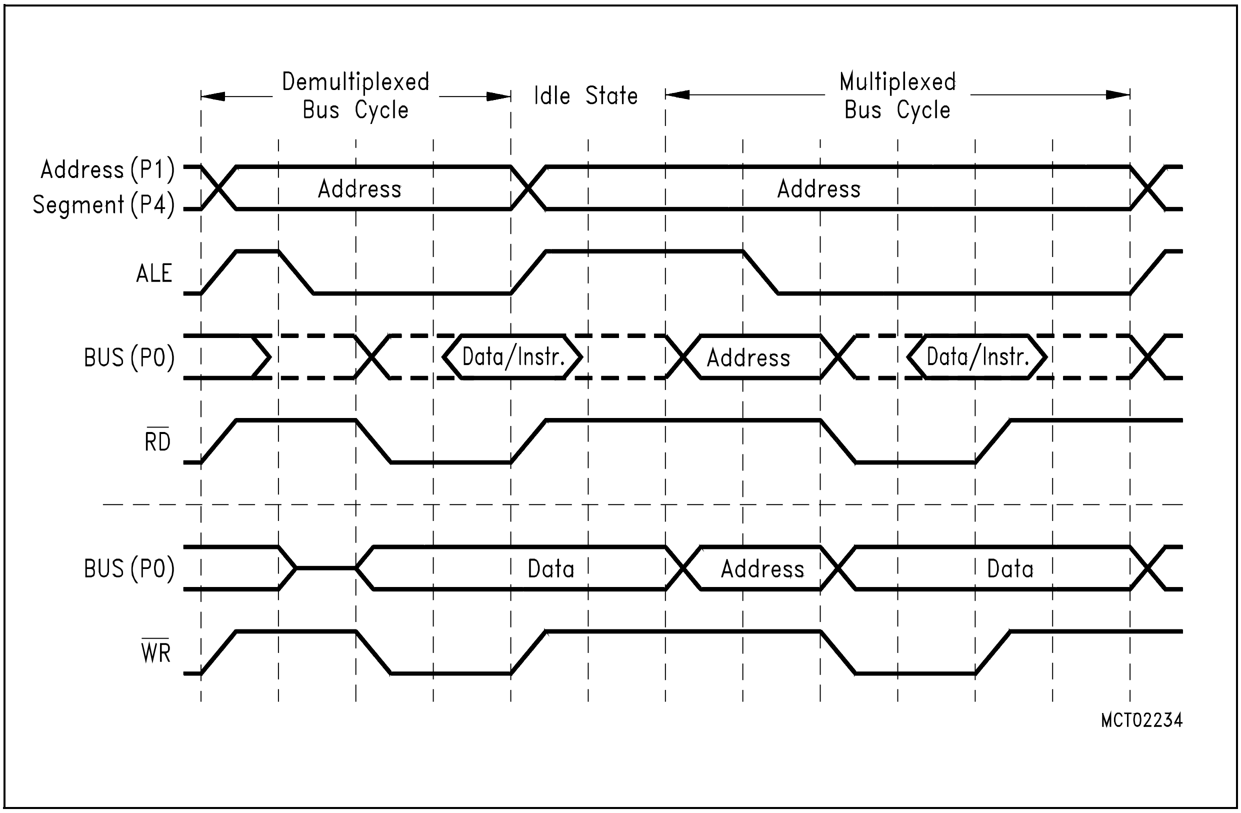


Figure 9-4 Switching from Demultiplexed to Multiplexed Bus Mode

Switching between external resources (e.g. different peripherals) may incur a problem if the previously accessed resource needs some time to switch of its output drivers (after a read) and the resource to be accessed next switches on its output drivers very fast. In systems running on higher frequencies this may lead to a bus conflict (the switch off delays normally are independent from the clock frequency).

In such a case an additional waitstate can automatically be inserted when leaving a certain address window, i.e. when the next cycle accesses a different window. This waitstate is controlled in the same way as the waitstate when switching from demultiplexed to multiplexed bus mode, see figure above.

BUSCON switch waitstates are enabled via bits BSWCx in the BUSCON registers. By enabling the automatic BUSCON switch waitstate (BSWCx = ’1’) there is no impact on the system performance as long as the external bus cycles access the same address window. Only if the following cycle accesses a different window a waitstate is inserted between the last access to the previous window and the first access to the new window.

After reset no BUSCON switch waitstates are selected.

External Data Bus Width

The EBC can operate on 8-bit or 16-bit wide external memory/peripherals. A 16-bit data bus uses PORT0, while an 8-bit data bus only uses P0L, the lower byte of PORT0. This saves on address latches, bus transceivers, bus routing and memory cost on the expense of transfer time. The EBC can control word accesses on an 8-bit data bus as well as byte accesses on a 16-bit data bus.

Word accesses on an 8-bit data bus are automatically split into two subsequent byte accesses, where the low byte is accessed first, then the high byte. The assembly of bytes to words and the disassembly of words into bytes is handled by the EBC and is transparent to the CPU and the programmer.

Byte accesses on a 16-bit data bus require that the upper and lower half of the memory can be accessed individually. In this case the upper byte is selected with the BHE signal, while the lower byte is selected with the A0 signal. So the two bytes of the memory can be enabled independent from each other, or together when accessing words.

When writing bytes to an external 16-bit device, which has a single CS input, but two WR enable inputs (for the two bytes), the EBC can directly generate these two write control signals. This saves the external combination of the WR signal with A0 or BHE. In this case pin WR serves as WRL (write low byte) and pin BHE serves as WRH (write high byte). Bit WRCFG in register SYSCON selects the operating mode for pins WR and BHE. The respective byte will be written on both data bus halfs.

When reading bytes from an external 16-bit device, whole words may be read and the C164 automatically selects the byte to be input and discards the other. However, care must be taken when reading devices that change state when being read, like FIFOs, interrupt status registers, etc. In this case individual bytes should be selected using BHE and A0.

Table 9-2 Bus Mode Versus Performance

Bus Mode	**Transfer Rate** **(**Speed factor for byte/word/dword access**)**	**System Requirements**	**Free IO Lines**
8-bit Multiplexed	Very low (1.5 / 3 / 6)	Low (8-bit latch, byte bus)	P1H, P1L
8-bit Demultipl.	Low (1 / 2 / 4)	Very low (no latch, byte bus)	P0H
16-bit Multiplexed	High (1.5 / 1.5 / 3)	High (16-bit latch, word bus)	P1H, P1L
16-bit Demultipl.	Very high (1 / 1 / 2)	Low (no latch, word bus)	---

Note: PORT1 becomes available for general purpose IO, when none of the BUSCON registers selects a demultiplexed bus mode.

Disable/Enable Control for Pin $\overline{\text{BHE}}$ (BYTDIS)

Bit BYTDIS is provided for controlling the active low Byte High Enable ($\overline{\text{BHE}}$) pin. The function of the $\overline{\text{BHE}}$ pin is enabled, if the BYTDIS bit contains a ’0’. Otherwise, it is disabled and the pin can be used as standard IO pin. The $\overline{\text{BHE}}$ pin is implicitly used by the External Bus Controller to select one of two byte-organized memory chips, which are connected to the C164 via a word-wide external data bus. After reset the $\overline{\text{BHE}}$ function is automatically enabled (BYTDIS = ’0’), if a 16-bit data bus is selected during reset, otherwise it is disabled (BYTDIS=’1’). It may be disabled, if byte access to 16-bit memory is not required, and the $\overline{\text{BHE}}$ signal is not used.

Segment Address Generation

During external accesses the EBC generates a (programmable) number of address lines on Port 4, which extend the 16-bit address output on PORT0 and so increase the accessible address space. The number of segment address lines is selected during reset and coded in bit field SALSEL in register RP0H (see table below).

Table 9-3 Decoding of Segment Address Lines

SALSEL	Segment Address Lines		Directly accessible Address Space	
1 1	Two:	A17...A16	256	KByte (Default without pull-downs)
1 0	Six:	A21...A16	4	MByte (Maximum)
0 1	None		64	KByte (Minimum)
0 0	Four:	A19...A16	1	MByte

Note: The total accessible address space may be increased by accessing several banks which are distinguished by individual chip select lines.

If Port 4 is used to output segment address lines, in most cases the drivers must operate in push/pull mode. Make sure that OPD4 does not select open drain mode in this case.

CS Signal Generation

During external accesses the EBC can generate a (programmable) number of CS lines on Port 4, which allow to directly select external peripherals or memory banks without requiring an external decoder. The number of CS lines is selected during reset and coded in bit field CSSEL in register RP0H (see table below).

Table 9-4 Decoding of Chip Select Lines

CSSEL	Chip Select Lines	Note
1 1	Four: CS3...CS0	Default without pull-downs
1 0	None	
0 1	Two: CS1...CS0	
0 0	Three: CS2...CS0	

The CSx outputs are associated with the BUSCONx registers and are driven active (low) for any access within the address area defined for the respective BUSCON register. For any access outside this defined address area the respective CSx signal will go inactive (high). At the beginning of each external bus cycle the corresponding valid CS signal is determined and activated. All other CS lines are deactivated (driven high) at the same time.

Note: The CSx signals will not be updated for an access to any internal address area (i.e. when no external bus cycle is started), even if this area is covered by the respective ADDRSELx register. An access to an on-chip X-Peripheral deactivates all external CS signals.
Upon accesses to address windows without a selected CS line all selected CS lines are deactivated.

The chip select signals allow to be operated in four different modes (see table below) which are selected via bits CSWENx and CSRENx in the respective BUSCONx register.

Table 9-5 Chip Select Generation Modes

CSWENx	CSRENx	Chip Select Mode
0	0	Address Chip Select (Default after Reset)
0	1	Read Chip Select
1	0	Write Chip Select
1	1	Read/Write Chip Select

Read or Write Chip Select signals remain active only as long as the associated control signal ($\overline{\text{RD}}$ or $\overline{\text{WR}}$) is active. This also includes the programmable read/write delay. Read chip select is only activated for read cycles, write chip select is only activated for write cycles, read/write chip select is activated for both read and write cycles (write cycles are assumed, if any of the signals $\overline{\text{WRH}}$ or $\overline{\text{WRL}}$ gets active). These modes save external glue logic, when accessing external devices like latches or drivers that only provide a single enable input.

Address Chip Select signals remain active during the complete bus cycle. For address chip select signals two generation modes can be selected via bit CSCFG in register SYSCON:
- A **latched** address chip select signal (CSCFG='0') becomes active with the falling edge of ALE and becomes inactive at the beginning of an external bus cycle that accesses a different address window. No spikes will be generated on the chip select lines and no changes occur as long as locations within the same address window or within internal memory (excluding X-Peripherals and XRAM) are accessed.
- An **early** address chip select signal (CSCFG='1') becomes active together with the address and $\overline{\text{BHE}}$ (if enabled) and remains active until the end of the current bus cycle. Early address chip select signals are not latched internally and may toggle intermediately while the address is changing.

Note: $\overline{\text{CS0}}$ provides a latched address chip select directly after reset (except for single chip mode) when the first instruction is fetched.

Internal pullup devices hold all $\overline{\text{CS}}$ lines high during reset. After the end of a reset sequence the pullup devices are switched off and the pin drivers control the pin levels on the selected $\overline{\text{CS}}$ lines. Not selected $\overline{\text{CS}}$ lines will enter the high-impedance state and are available for general purpose IO.

Segment Address versus Chip Select

The external bus interface of the C164 supports many configurations for the external memory. By increasing the number of segment address lines the C164 can address a linear address space of 256 KByte, 1 MByte or 4 MByte. This allows to implement a large sequential memory area, and also allows to access a great number of external devices, using an external decoder. By increasing the number of $\overline{\text{CS}}$ lines the C164 can access memory banks or peripherals without external glue logic. These two features may be combined to optimize the overall system performance.

Note: If the number of segment address lines and $\overline{\text{CS}}$ lines configured at reset cause overlap (eg. A17...A16 ***and*** *$\overline{\text{CS3}}$...$\overline{\text{CS0}}$) then the segment address line function will take precedence. In this example the segment address lines (A16 and A17) will be available but only 2 chip select lines ($\overline{\text{CS0}}$ and $\overline{\text{CS1}}$) will be available.*
Bit SGTDIS of register SYSCON defines, if the CSP register is saved during interrupt entry (segmentation active) or not (segmentation disabled).

9.3 Programmable Bus Characteristics

Important timing characteristics of the external bus interface have been made user programmable to allow to adapt it to a wide range of different external bus and memory configurations with different types of memories and/or peripherals.

The following parameters of an external bus cycle are programmable:

- **ALE Control** defines the ALE signal length and the address hold time after its falling edge
- **Memory Cycle Time** (extendable with 1...15 waitstates) defines the allowable access time
- **Memory Tri-State Time** (extendable with 1 waitstate) defines the time for a data driver to float
- **Read/Write Delay Time** defines when a command is activated after the falling edge of ALE

Note: Internal accesses are executed with maximum speed and therefore are not programmable.
External accesses use the slowest possible bus cycle after reset. The bus cycle timing may then be optimized by the initialization software.

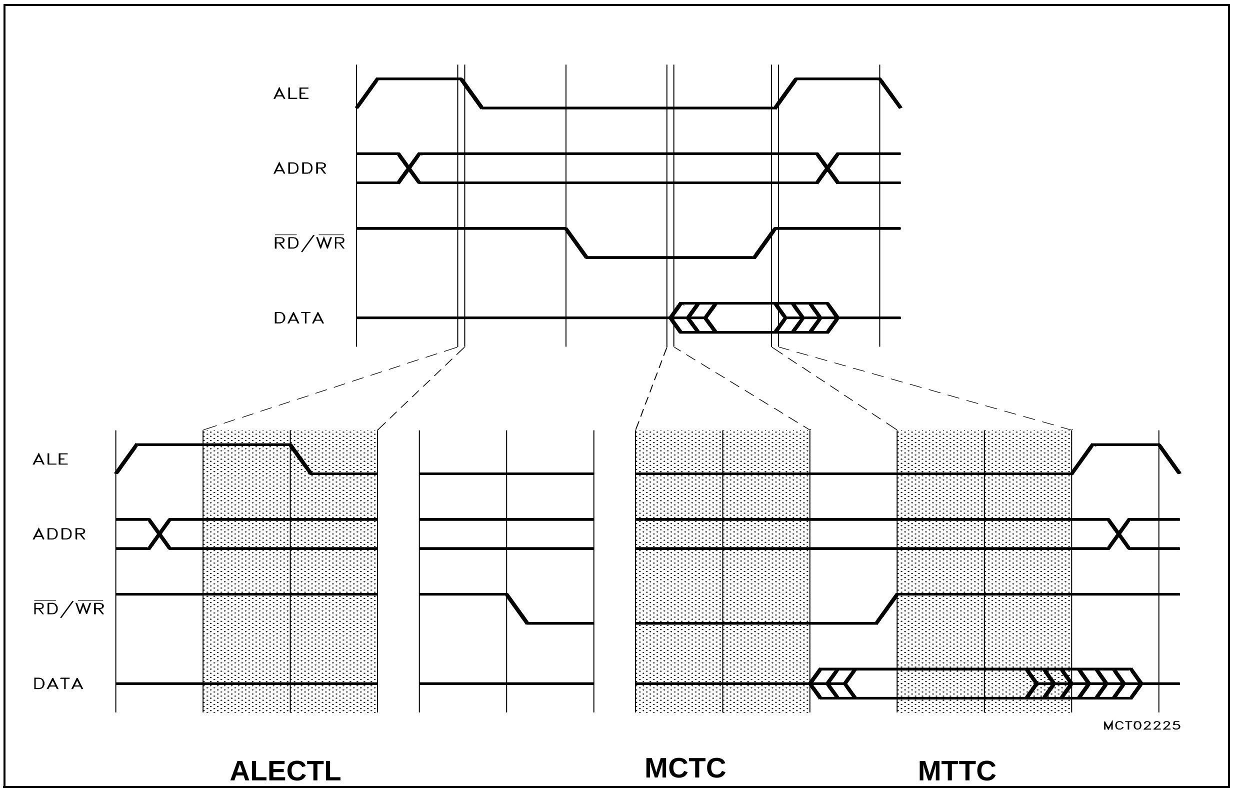


Figure 9-5 Programmable External Bus Cycle

ALE Length Control

The length of the ALE signal and the address hold time after its falling edge are controlled by the ALECTLx bits in the BUSCON registers. When bit ALECTL is set to ‘1’, external bus cycles accessing the respective address window will have their ALE signal prolonged by half a CPU clock (1 TCL). Also the address hold time after the falling edge of ALE (on a multiplexed bus) will be prolonged by half a CPU clock, so the data transfer within a bus cycle refers to the same CLKOUT edges as usual (i.e. the data transfer is delayed by one CPU clock). This allows more time for the address to be latched.

Note: ALECTL0 is ‘1’ after reset to select the slowest possible bus cycle, the other ALECTLx are ‘0’ after reset.

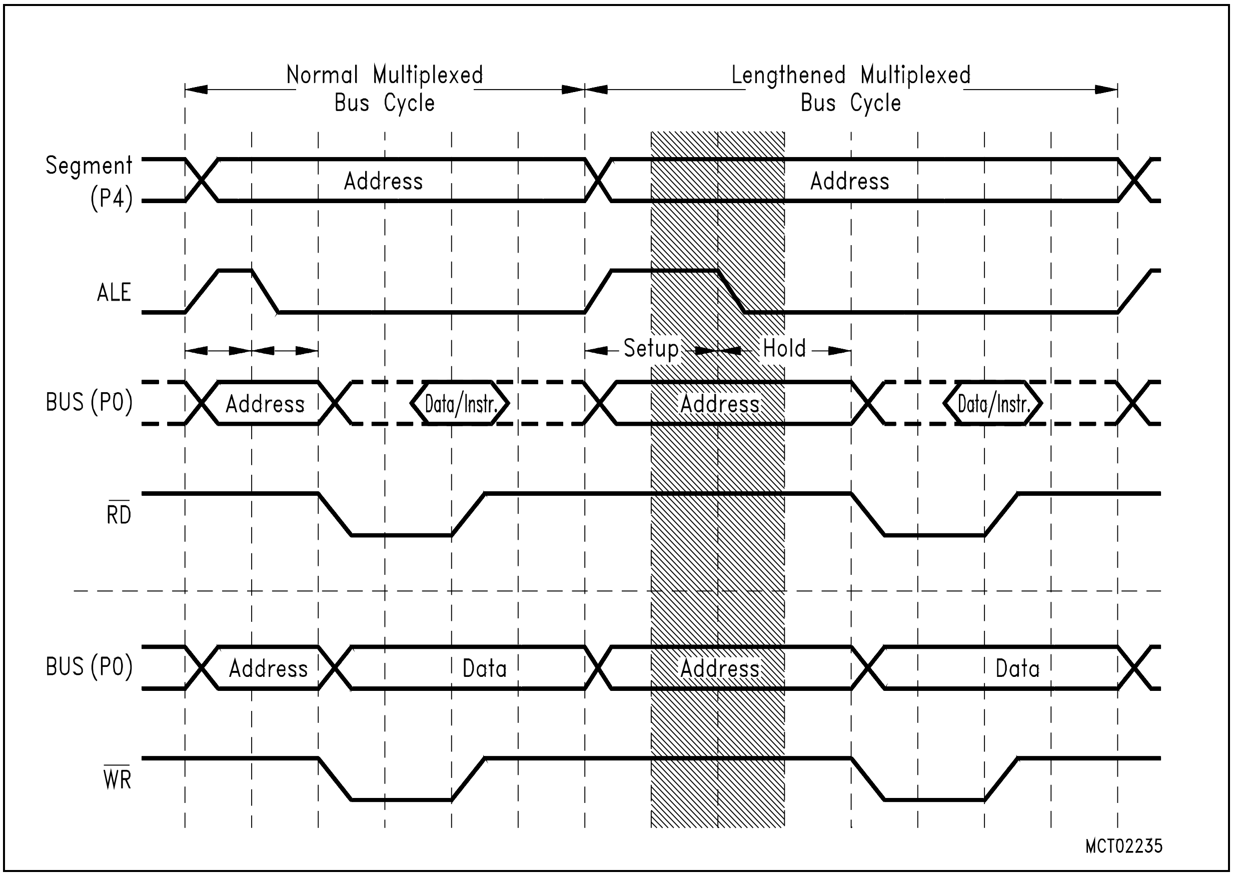


Figure 9-6 ALE Length Control

Programmable Memory Cycle Time

The C164 allows the user to adjust the controller's external bus cycles to the access time of the respective memory or peripheral. This access time is the total time required to move the data to the destination. It represents the period of time during which the controller's signals do not change.

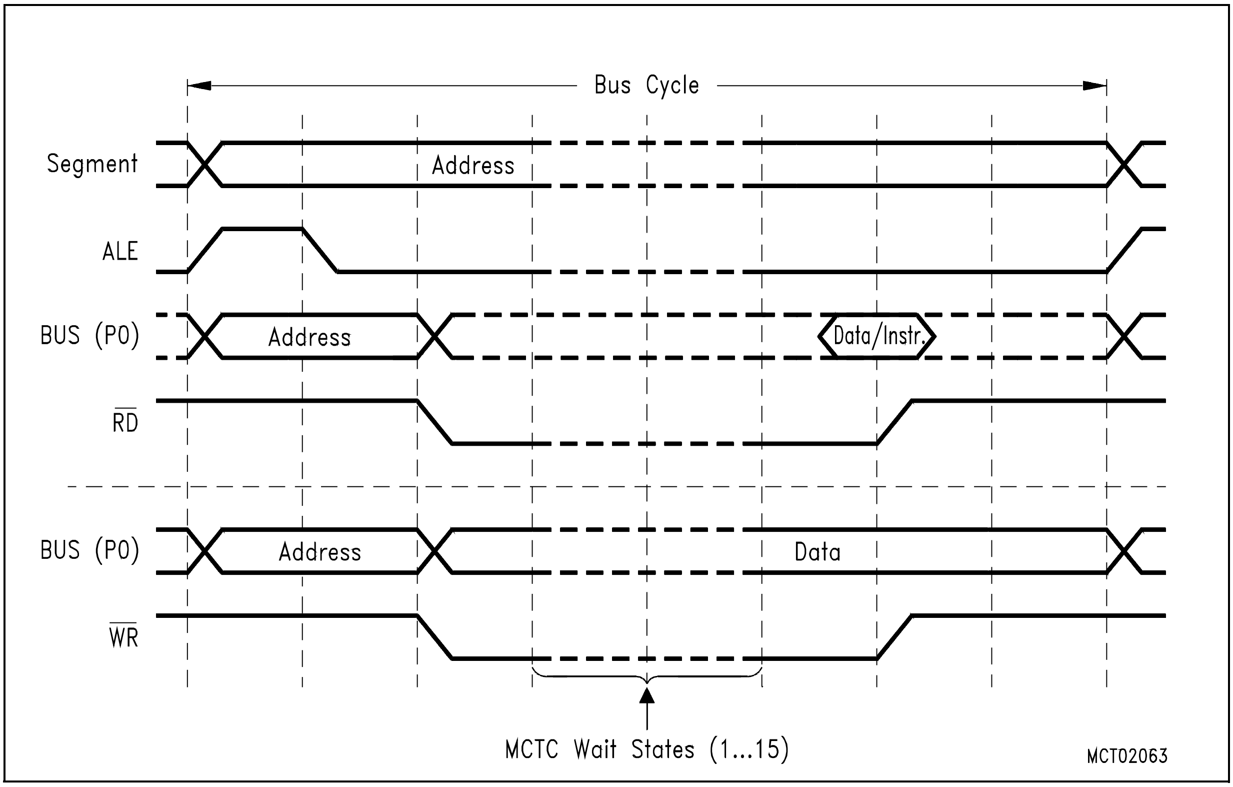


Figure 9-7 Memory Cycle Time

The external bus cycles of the C164 can be extended for a memory or peripheral, which cannot keep pace with the controller's maximum speed, by introducing wait states during the access (see figure above). During these memory cycle time wait states, the CPU is idle, if this access is required for the execution of the current instruction.

The memory cycle time wait states can be programmed in increments of one CPU clock (2 TCL) within a range from 0 to 15 (default after reset) via the MCTC fields of the BUSCON registers. 15-<MCTC> waitstates will be inserted.

Programmable Memory Tri-State Time

The C164 allows the user to adjust the time between two subsequent external accesses to account for the tri-state time of the external device. The tri-state time defines, when the external device has released the bus after deactivation of the read command ($\overline{RD}$).

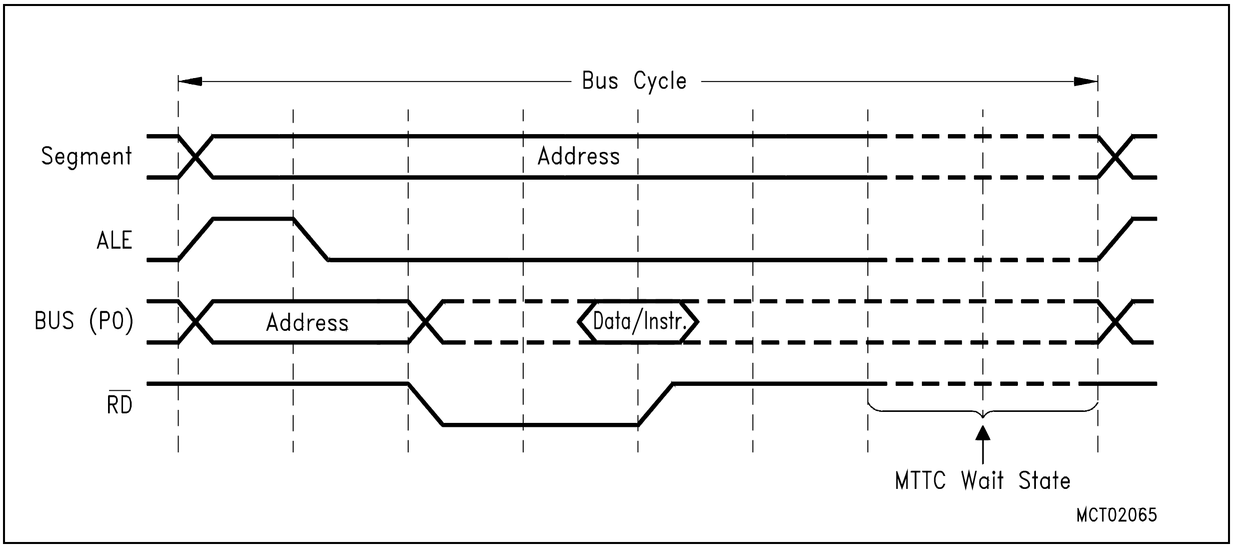


Figure 9-8 Memory Tri-State Time

The output of the next address on the external bus can be delayed for a memory or peripheral, which needs more time to switch off its bus drivers, by introducing a wait state after the previous bus cycle (see figure above). During this memory tri-state time wait state, the CPU is not idle, so CPU operations will only be slowed down if a subsequent external instruction or data fetch operation is required during the next instruction cycle.

The memory tri-state time waitstate requires one CPU clock (2 TCL) and is controlled via the MTTCx bits of the BUSCON registers. A waitstate will be inserted, if bit MTTCx is ‘0’ (default after reset).

Note: External bus cycles in multiplexed bus modes implicitly add one tri-state time waitstate in addition to the programmable MTTC waitstate.

Read/Write Signal Delay

The C164 allows the user to adjust the timing of the read and write commands to account for timing requirements of external peripherals. The read/write delay controls the time between the falling edge of ALE and the falling edge of the command. Without read/write delay the falling edges of ALE and command(s) are coincident (except for propagation delays). With the delay enabled, the command(s) become active half a CPU clock (1 TCL) after the falling edge of ALE.
The read/write delay does not extend the memory cycle time, and does not slow down the controller in general. In multiplexed bus modes, however, the data drivers of an external device may conflict with the C164's address, when the early $\overline{\text{RD}}$ signal is used. Therefore multiplexed bus cycles should always be programmed with read/write delay.

The read/write delay is controlled via the RWDCx bits in the BUSCON registers. The command(s) will be delayed, if bit RWDCx is '0' (default after reset).

Early $\overline{\text{WR}}$ Signal Deactivation

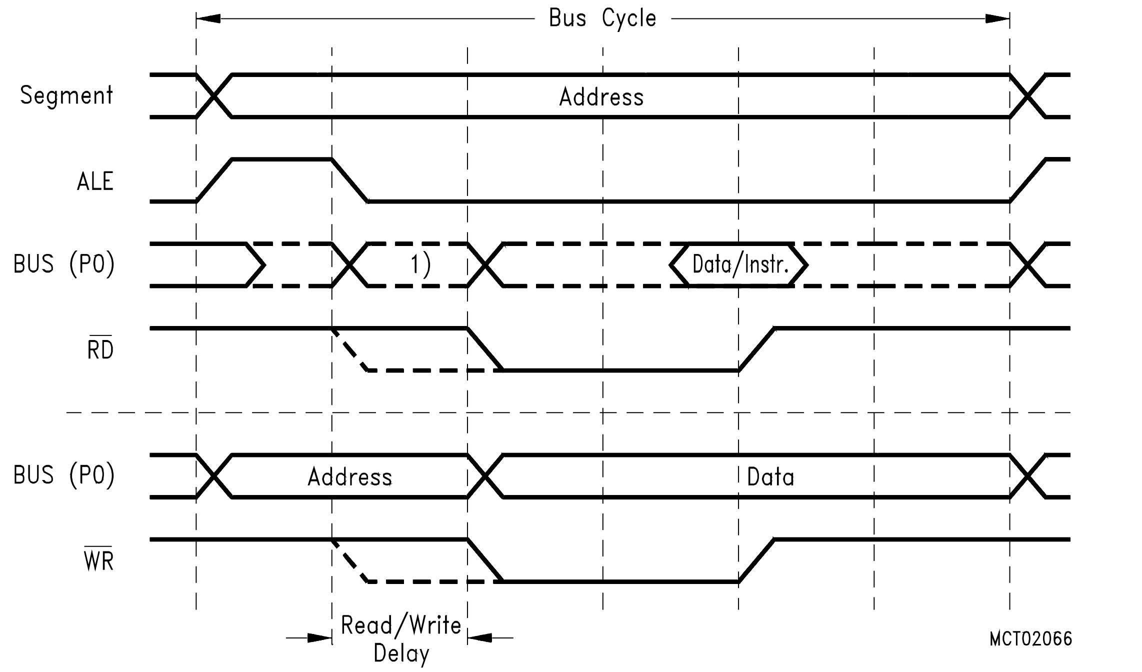
The duration of an external write access can be shortened by one TCL. The $\overline{\text{WR}}$ signal is activated (driven low) in the standard way, but can be deactivated (driven high) one TCL earlier than defined in the standard timing. In this case also the data output drivers will be deactivated one TCL earlier.

This is especially useful in systems which operate on higher CPU clock frequencies and employ external modules (memories, peripherals, etc.) which switch on their own data drivers very fast in response to e.g. a chip select signal.

Conflicts between the C164's and the external peripheral's output drivers can be avoided then by selecting early $\overline{\text{WR}}$ for the C164.

Note: Make sure that the reduced $\overline{\text{WR}}$ low time then still matches the requirements of the external peripheral/memory.

Early $\overline{\text{WR}}$ deactivation is controlled via the EWENx bits in the BUSCON registers. The $\overline{\text{WR}}$ signal will be shortened if bit EWENx is '1' (default after reset is a standard $\overline{\text{WR}}$ signal, i.e. EWENx = '0').



1) The data drivers from the previous bus cycle should be disabled when the $\overline{RD}$ signal becomes active.

Figure 9-9 Read/Write Signal Duration Control

9.4 Controlling the External Bus Controller

A set of registers controls the functions of the EBC. General features like the usage of interface pins ($\overline{\text{WR}}$, $\overline{\text{BHE}}$), segmentation and internal ROM mapping are controlled via register SYSCON. The properties of a bus cycle like chip select mode, length of ALE, external bus mode, read/write delay and waitstates are controlled via registers BUSCON4...BUSCON0. Four of these registers (BUSCON4...BUSCON1) have an address select register (ADDRSEL4...ADDRSEL1) associated with them, which allows to specify up to four address areas and the individual bus characteristics within these areas. All accesses that are not covered by these four areas are then controlled via BUSCON0. This allows to use memory components or peripherals with different interfaces within the same system, while optimizing accesses to each of them.

SYSCON

System Control Register **SFR (FF12$_H$/89$_H$)** **Reset value: 0XX0$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STKSZ			ROM S1	SGT DIS	ROM EN	BYT DIS	CLK EN	WR CFG	CS CFG	-	OWD DIS	BD RST EN	XPEN	VISI-BLE	-
rw			rw	rw	rwh	rwh	rw	rwh	rw	-	rwh	rw	rw	rw	-

Bit	Function
VISIBLE	**Visible Mode Control** 0: Accesses to XBUS peripherals are done internally 1: XBUS peripheral accesses are made visible on the external pins
XPEN	**XBUS Peripheral Enable Bit** 0: Accesses to the on-chip X-Peripherals and their functions are disabled 1: The on-chip X-Peripherals are enabled and can be accessed
BDRSTEN	**Bidirectional Reset Enable Bit** 0: Pin $\overline{\text{RSTIN}}$ is an input only. 1: Pin $\overline{\text{RSTIN}}$ is pulled low during the internal reset sequence after any reset.
OWDDIS	**Oscillator Watchdog Disable Bit** 0: The on-chip oscillator watchdog is enabled and active. 1: The on-chip oscillator watchdog is disabled and the CPU clock is always fed from the oscillator input.
CSCFG	**Chip Select Configuration Control** 0: Latched $\overline{\text{CS}}$ mode. The $\overline{\text{CS}}$ signals are latched internally and driven to the (enabled) port pins synchronously. 1: Unlatched $\overline{\text{CS}}$ mode. The $\overline{\text{CS}}$ signals are directly derived from the address and driven to the (enabled) port pins.

Bit	Function
WRCFG	**Write Configuration Control** (Set according to pin P0H.0 during reset) 0: Pins $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ retain their normal function 1: Pin $\overline{\text{WR}}$ acts as $\overline{\text{WRL}}$, pin $\overline{\text{BHE}}$ acts as $\overline{\text{WRH}}$
CLKEN	**System Clock Output Enable** (CLKOUT) 0: CLKOUT disabled: pin may be used for general purpose IO or for signal FOUT 1: CLKOUT enabled: pin outputs the system clock signal
BYTDIS	**Disable/Enable Control for Pin $\overline{\text{BHE}}$** (Set according to data bus width) 0: Pin $\overline{\text{BHE}}$ enabled 1: Pin $\overline{\text{BHE}}$ disabled, pin may be used for general purpose IO
ROMEN	**Internal ROM Enable** (Set according to pin $\overline{\text{EA}}$ during reset) 0: Internal program memory disabled, accesses to the ROM area use the external bus 1: Internal program memory enabled
SGTDIS	**Segmentation Disable/Enable Control** 0: Segmentation enabled (CSP is saved/restored during interrupt entry/exit) 1: Segmentation disabled (Only IP is saved/restored)
ROMS1	**Internal ROM Mapping** 0: Internal ROM area mapped to segment 0 (00’0000$_H$...00’7FFF$_H$) 1: Internal ROM area mapped to segment 1 (01’0000$_H$...01’7FFF$_H$)
STKSZ	**System Stack Size** Selects the size of the system stack (in the internal RAM) from 32 to 1024 words

Note: Register SYSCON cannot be changed after execution of the EINIT instruction. Bit SGTDIS controls the correct stack operation (push/pop of CSP or not) during traps and interrupts.

The layout of the BUSCON registers and ADDRSEL registers is identical.

Registers BUSCON4...BUSCON1, which control the selected address windows, are completely under software control, while register BUSCON0, which e.g. is also used for the very first code access after reset, is partly controlled by hardware, i.e. it is initialized via PORT0 during the reset sequence. This hardware control allows to define an appropriate external bus for systems, where no internal program memory is provided.

BUSCON0
Bus Control Register 0 SFR (FF0C$_H$/86$_H$) Reset value: 0XX0$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN0	CSREN0	-	-	BSWC0	BUSACT0	ALECTL0	EWEN0	BTYP		MTTC0	RWDC0	MCTC			
rw	rw	-	-	rw	rwh	rwh	rw	rwh		rw	rw	rw			

BUSCON1
Bus Control Register 1 SFR (FF14$_H$/8A$_H$) Reset value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN1	CSREN1	-	-	BSWC1	BUSACT1	ALECTL1	EWEN1	BTYP		MTTC1	RWDC1	MCTC			
rw	rw	-	-	rw	rw	rw	rw	rw		rw	rw	rw			

BUSCON2
Bus Control Register 2 SFR (FF16$_H$/8B$_H$) Reset value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN2	CSREN2	-	-	BSWC2	BUSACT2	ALECTL2	EWEN2	BTYP		MTTC2	RWDC2	MCTC			
rw	rw	-	-	rw	rw	rw	rw	rw		rw	rw	rw			

BUSCON3
Bus Control Register 3 SFR (FF18$_H$/8C$_H$) Reset value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN3	CSREN3	-	-	BSWC3	BUSACT3	ALECTL3	EWEN3	BTYP		MTTC3	RWDC3	MCTC			
rw	rw	-	-	rw	rw	rw	rw	rw		rw	rw	rw			

BUSCON4
Bus Control Register 4 SFR (FF1A$_H$/8D$_H$) Reset value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN4	CSREN4	-	-	BSWC4	BUSACT4	ALECTL4	EWEN4	BTYP		MTTC4	RWDC4	MCTC			
rw	rw	-	-	rw	rw	rw	rw	rw		rw	rw	rw			

Note: BUSCON0 is initialized with 00C0$_H$, if pin $\overline{EA}$ is high during reset. If pin $\overline{EA}$ is low during reset, bits BUSACT0 and ALECTL0 are set ('1') and bit field BTYP is loaded with the bus configuration selected via PORT0.

Bit	Function
MCTC	**Memory Cycle Time Control** (Number of memory cycle time wait states) 0000: 15 waitstates . . . (Number = 15 - <MCTC>) 1111: No waitstates
RWDCx	**Read/Write Delay Control for BUSCONx** 0: With rd/wr delay: activate command 1 TCL after falling edge of ALE 1: No rd/wr delay: activate command with falling edge of ALE
MTTCx	**Memory Tristate Time Control** 0: 1 waitstate 1: No waitstate
BTYP	**External Bus Configuration** 00: 8-bit Demultiplexed Bus 01: 8-bit Multiplexed Bus 10: 16-bit Demultiplexed Bus 11: 16-bit Multiplexed Bus *Note: For BUSCON0 BTYP is defined via PORT0 during reset.*
EWENx	**Early Write Enable** 0: Normal $\overline{WR}$ signal 1: Early write: The $\overline{WR}$ signal is deactivated and write data is tristated one TCL earlier
ALECTLx	**ALE Lengthening Control** 0: Normal ALE signal 1: Lengthened ALE signal
BUSACTx	**Bus Active Control** 0: External bus disabled 1: External bus enabled within respective address window (ADDRSEL)
BSWCx	**BUSCON Switch Control** 0: Address windows are switched immediately 1: A tristate waitstate is inserted if the next bus cycle accesses a different window than the one controlled by this BUSCON register.[1]
CSRENx	**Read Chip Select Enable** 0: The $\overline{CS}$ signal is independent of the read command ($\overline{RD}$) 1: The $\overline{CS}$ signal is generated for the duration of the read command
CSWENx	**Write Chip Select Enable** 0: The $\overline{CS}$ signal is independent of the write cmd. ($\overline{WR}$,$\overline{WRL}$,$\overline{WRH}$) 1: The $\overline{CS}$ signal is generated for the duration of the write command

[1] A BUSCON switch waitstate is enabled by bit BUSCONx.BSWCx of the address window that is left.

ADDRSEL1

Address Select Register 1 SFR (FE18$_H$/0C$_H$) Reset value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RGSAD												RGSZ			
rw												rw			

ADDRSEL2

Address Select Register 2 SFR (FE1A$_H$/0D$_H$) Reset value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RGSAD												RGSZ			
rw												rw			

ADDRSEL3

Address Select Register 3 SFR (FE1C$_H$/0E$_H$) Reset value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RGSAD												RGSZ			
rw												rw			

ADDRSEL4

Address Select Register 4 SFR (FE1E$_H$/0F$_H$) Reset value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RGSAD												RGSZ			
rw												rw			

Bit	Function
RGSZ	**Range Size Selection** Defines the size of the address area controlled by the respective BUSCONx/ADDRSELx register pair. See table below.
RGSAD	**Range Start Address** Defines the upper bits of the start address of the respective address area. See table below.

Note: There is no register ADDRSEL0, as register BUSCON0 controls all external accesses outside the four address windows of BUSCON4...BUSCON1 within the complete address space.

Definition of Address Areas

The four register pairs BUSCON4/ADDRSEL4...BUSCON1/ADDRSEL1 allow to define 4 separate address areas within the address space of the C164. Within each of these address areas external accesses can be controlled by one of the four different bus modes, independent of each other and of the bus mode specified in register BUSCON0. Each ADDRSELx register in a way cuts out an address window, within which the parameters in register BUSCONx are used to control external accesses. The range start address of such a window defines the upper address bits, which are not used within the address window of the specified size (see table below). For a given window size only those upper address bits of the start address are used (marked “R”), which are not implicitly used for addresses inside the window. The lower bits of the start address (marked “x”) are disregarded.

Table 9-6 Address Window Definition

Bit field RGSZ	Resulting Window Size	Relevant Bits (R) of Start Addr. (A12...)											
0 0 0 0	4 KByte	R	R	R	R	R	R	R	R	R	R	R	R
0 0 0 1	8 KByte	R	R	R	R	R	R	R	R	R	R	R	x
0 0 1 0	16 KByte	R	R	R	R	R	R	R	R	R	R	x	x
0 0 1 1	32 KByte	R	R	R	R	R	R	R	R	R	x	x	x
0 1 0 0	64 KByte	R	R	R	R	R	R	R	R	x	x	x	x
0 1 0 1	128 KByte	R	R	R	R	R	R	R	x	x	x	x	x
0 1 1 0	256 KByte	R	R	R	R	R	R	x	x	x	x	x	x
0 1 1 1	512 KByte	R	R	R	R	R	x	x	x	x	x	x	x
1 0 0 0	1 MByte	R	R	R	R	x	x	x	x	x	x	x	x
1 0 0 1	2 MByte	R	R	R	x	x	x	x	x	x	x	x	x
1 0 1 0	4 MByte	R	R	x	x	x	x	x	x	x	x	x	x
1 0 1 1	8 MByte	R	x	x	x	x	x	x	x	x	x	x	x
1 1 x x	Reserved.												

Address Window Arbitration

The address windows that can be defined within the C164's address space may partly overlap each other. Thus e.g. small areas may be cut out of bigger windows in order to effectively utilize external resources, especially within segment 0.

For each access the EBC compares the current address with all address select registers (programmable ADDRSELx and hardwired XADRSx). This comparison is done in four levels.

Priority 1:The hardwired XADRSx registers are evaluated first. A match with one of these registers directs the access to the respective X-Peripheral using the corresponding XBCONx register and ignoring all other ADDRSELx registers.
Priority 2:Registers ADDRSEL2 and ADDRSEL4 are evaluated before ADDRSEL1 and ADDRSEL3, respectively. A match with one of these registers directs the access to the respective external area using the corresponding BUSCONx register and ignoring registers ADDRSEL1/3 (see figure below).
Priority 3:A match with registers ADDRSEL1 or ADDRSEL3 directs the access to the respective external area using the corresponding BUSCONx register.
Priority 4:If there is no match with any XADRSx or ADDRSELx register the access to the external bus uses register BUSCON0.

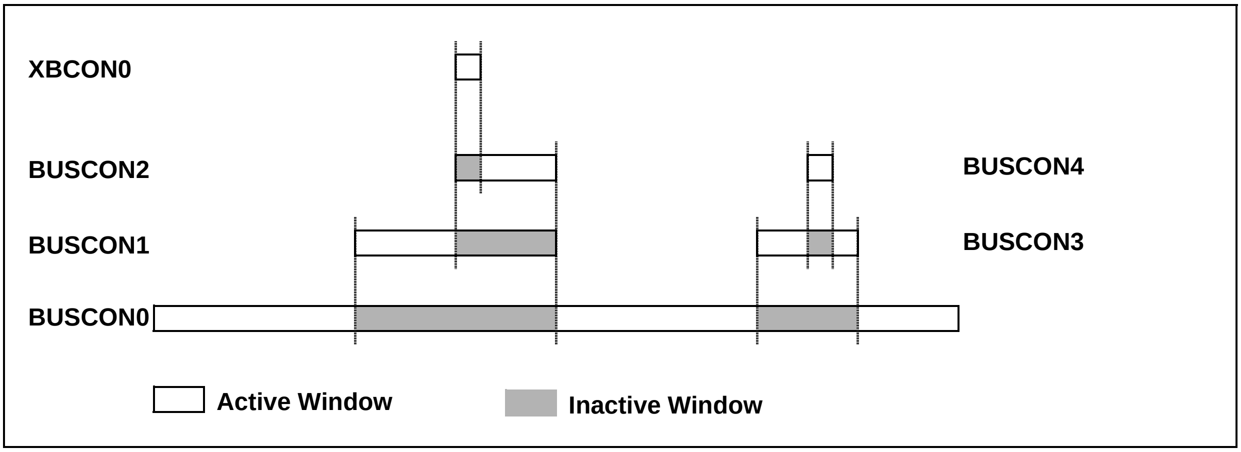


Figure 9-10 Address Window Arbitration

Note: Only the indicated overlaps are defined. All other overlaps lead to erroneous bus cycles. E.g. ADDRSEL4 may not overlap ADDRSEL2 or ADDRSEL1. The hardwired XADRSx registers are defined non-overlapping.

RP0H

Reset Value of P0H **SFR (F108$_H$/84$_H$)** **Reset value: - - XX$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								CLKCFG			**SALSEL**		**CSSEL**		**WRC**
								rh			rh		rh		rh

Bit	Function
WRC	**Write Configuration** 0: Pins $\overline{WR}$ and $\overline{BHE}$ operate as $\overline{WRL}$ and $\overline{WRH}$ signals 1: Pins $\overline{WR}$ and $\overline{BHE}$ operate as $\overline{WR}$ and $\overline{BHE}$ signals
CSSEL	**Chip Select Line Selection** (Number of active $\overline{CS}$ outputs) 00: 3 $\overline{CS}$ lines: $\overline{CS2}$...$\overline{CS0}$ 01: 2 $\overline{CS}$ lines: $\overline{CS1}$...$\overline{CS0}$ 10: No $\overline{CS}$ lines at all 11: 4 $\overline{CS}$ lines: $\overline{CS3}$...$\overline{CS0}$ (Default without pulldowns)
SALSEL	**Segment Address Line Selection** (Nr. of active segment addr. outputs) 00: 4-bit segment address: A19...A16 01: No segment address lines at all 10: 6-bit segment address: A21...A16 11: 2-bit segment address: A17...A16 (Default without pulldowns)
CLKCFG	**Clock Generation Mode Configuration** These pins define the clock generation mode, i.e. the mechanism how the internal CPU clock is generated from the externally applied (XTAL1) input clock.

Note: RP0H cannot be changed directly via software, but rather allows to check the current configuration.
Bitfields CLKCFG, SALSEL, and CSSEL may be reconfigured via register RSTCON.

Precautions and Hints

- The external bus interface is enabled as long as at least one of the BUSCON registers has its BUSACT bit set.
- PORT1 will output the intra-segment address as long as at least one of the BUSCON registers selects a demultiplexed external bus, even for multiplexed bus cycles.
- Not all address windows defined via registers ADDRSELx may overlap each other. The operation of the EBC will be erroneous in such a case. See chapter „Address Window Arbitration“.
- The address windows defined via registers ADDRSELx may overlap internal address areas. Internal accesses will be executed in this case.
- For any access to an internal address area the EBC will remain inactive (see EBC Idle State).

9.5 EBC Idle State

When the external bus interface is enabled, but no external access is currently executed, the EBC is idle. As long as only internal resources (from an architecture point of view) like IRAM, GPRs or SFRs, etc. are used the external bus interface does not change (see table below).

Accesses to on-chip X-Peripherals are also controlled by the EBC. However, even though an X-Peripheral appears like an external peripheral to the controller, the respective accesses do not generate valid external bus cycles.

Due to timing constraints address and write data of an XBUS cycle are reflected on the external bus interface (see table below). The „address" mentioned above includes Port 4, $\overline{\text{BHE}}$ and ALE which also pulses for an XBUS cycle. The external $\overline{\text{CS}}$ signals are driven inactive (high) because the EBC switches to an internal $\overline{\text{XCS}}$ signal.

The **external control signals** ($\overline{\text{RD}}$ and $\overline{\text{WR}}$ or $\overline{\text{WRL}}$/$\overline{\text{WRH}}$ if enabled) **remain inactive** (high).

Table 9-7 Status Of The External Bus Interface During EBC Idle State

Pins	Internal accesses only	XBUS accesses
PORT0	Tristated (floating)	Tristated (floating) for read accesses XBUS write data for write accesses
PORT1	Last used external address (if used for the bus interface)	Last used XBUS address (if used for the bus interface)
Port 4	Last used external segment address (on selected pins)	Last used XBUS segment address (on selected pins)
	Active external $\overline{\text{CS}}$ signal corresponding to last used address	Inactive (high) for selected $\overline{\text{CS}}$ signals
$\overline{\text{BHE}}$	Level corresponding to last external access	Level corresponding to last XBUS access
ALE	Inactive (low)	Pulses as defined for X-Peripheral
$\overline{\text{RD}}$	Inactive (high)	**Inactive (high)**
$\overline{\text{WR}}$/$\overline{\text{WRL}}$	Inactive (high)	**Inactive (high)**
$\overline{\text{WRH}}$	Inactive (high)	**Inactive (high)**

9.6 The XBUS Interface

The C164 provides an on-chip interface (the XBUS interface), via which integrated customer/application specific peripherals can be connected to the standard controller core. The XBUS is an internal representation of the external bus interface, i.e. it is operated in the same way.

For each peripheral on the XBUS (X-Peripheral) there is a separate address window controlled by a register pair XBCONx/XADRSx (similar to registers BUSCONx and ADDRSELx). As an interface to a peripheral in many cases is represented by just a few registers, the registers partly select smaller address windows than the standard ADDRSEL registers. As the register pairs control integrated peripherals rather than externally connected ones, most of them are fixed by mask programming rather than being user programmable.

X-Peripheral accesses provide the same choices as external accesses, so these peripherals may be bytewide or wordwide. Because the on-chip connection can be realized very efficient and for performance reasons X-Peripherals are only implemented with a separate address bus, i.e. in demultiplexed bus mode. Interrupt nodes are provided for X-Peripherals to be integrated.

Note: If you plan to develop a peripheral of your own to be integrated into a C164 device to create a customer specific version, please ask for the specification of the XBUS interface and for further support.

Enabling of XBUS Peripherals

After reset all on-chip XBUS peripherals are disabled. In order to be usable an XBUS peripheral must be enabled via the global enable bit XPEN in register SYSCON.

The following table summarizes the XBUS peripherals and also the number of waitstates which are used when accessing the respective peripheral.

Table 9-8 XBUS Peripherals in the C164

Associated XBUS Peripheral	Waitstates
CAN	2
XRAM 2 KByte	0
DataFlash/EEPROM 4 KByte [1]	0

[1] Available only in devices in Flash technology.

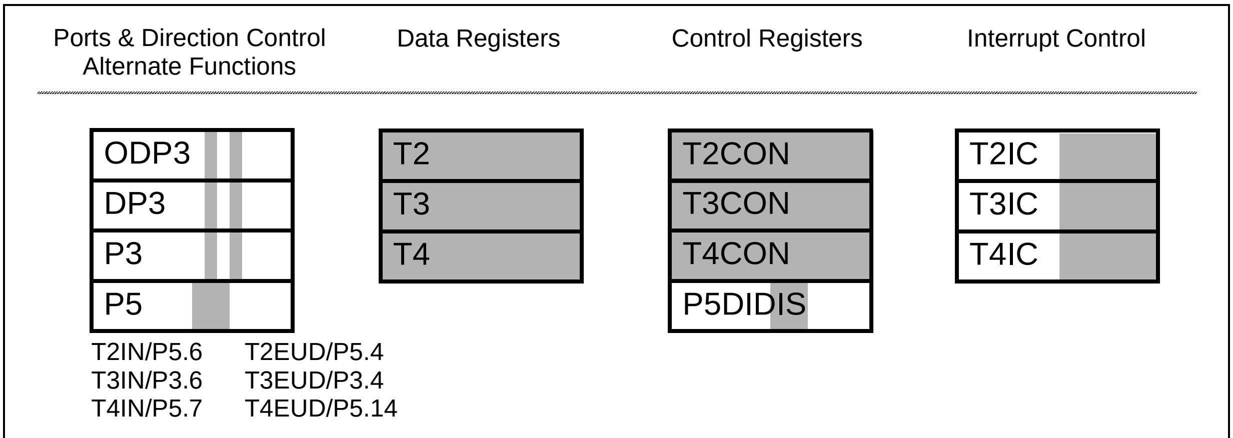
10 The General Purpose Timer Unit

The General Purpose Timer Unit GPT1 represents a very flexible multifunctional timer structure which may be used for timing, event counting, pulse width measurement, pulse generation, frequency multiplication, and other purposes.

Block GPT1 contains 3 timers/counters with a maximum resolution of 16 TCL. Each timer may operate independently in a number of different modes such as gated timer or counter mode, or may be concatenated with another timer of the same block. The auxiliary timers of GPT1 may optionally be configured as reload or capture registers for the core timer. GPT1 has alternate input/output functions and specific interrupts associated with it.

10.1 Timer Block GPT1

From a programmer's point of view, the GPT1 block is composed of a set of SFRs as summarized below. Those portions of port and direction registers which are used for alternate functions by the GPT1 block are shaded.



P5	Port 5 Data Register	P5DIDIS	Port 5 Digital Input Disable Register
ODP3	Port 3 Open Drain Control Register	T2	GPT1 Timer 2 Register
DP3	Port 3 Direction Control Register	T3	GPT1 Timer 3 Register
P3	Port 3 Data Register	T4	GPT1 Timer 4 Register
T2CON	GPT1 Timer 2 Control Register	T2IC	GPT1 Timer 2 Interrupt Control Register
T3CON	GPT1 Timer 3 Control Register	T3IC	GPT1 Timer 3 Interrupt Control Register
T4CON	GPT1 Timer 4 Control Register	T4IC	GPT1 Timer 4 Interrupt Control Register

Figure 10-1 SFRs and Port Pins Associated with Timer Block GPT1

All three timers of block GPT1 (T2, T3, T4) can run in 4 basic modes, which are timer, gated timer, counter and incremental interface mode, and all timers can either count up or down. Each timer has an alternate input function pin (TxIN) associated with it which serves as the gate control in gated timer mode, or as the count input in counter mode. The count direction (Up / Down) may be programmed via software or may be dynamically altered by a signal at an external control input pin.

Each overflow/underflow of core timer T3 is latched in the toggle FlipFlop T3OTL and may be indicated on an alternate output function pin. The auxiliary timers T2 and T4 may additionally be concatenated with the core timer, or used as capture or reload registers for the core timer.

The current contents of each timer can be read or modified by the CPU by accessing the corresponding timer registers T2, T3, or T4, which are located in the non-bitaddressable SFR space. When any of the timer registers is written to by the CPU in the state immediately before a timer increment, decrement, reload, or capture is to be performed, the CPU write operation has priority in order to guarantee correct results.

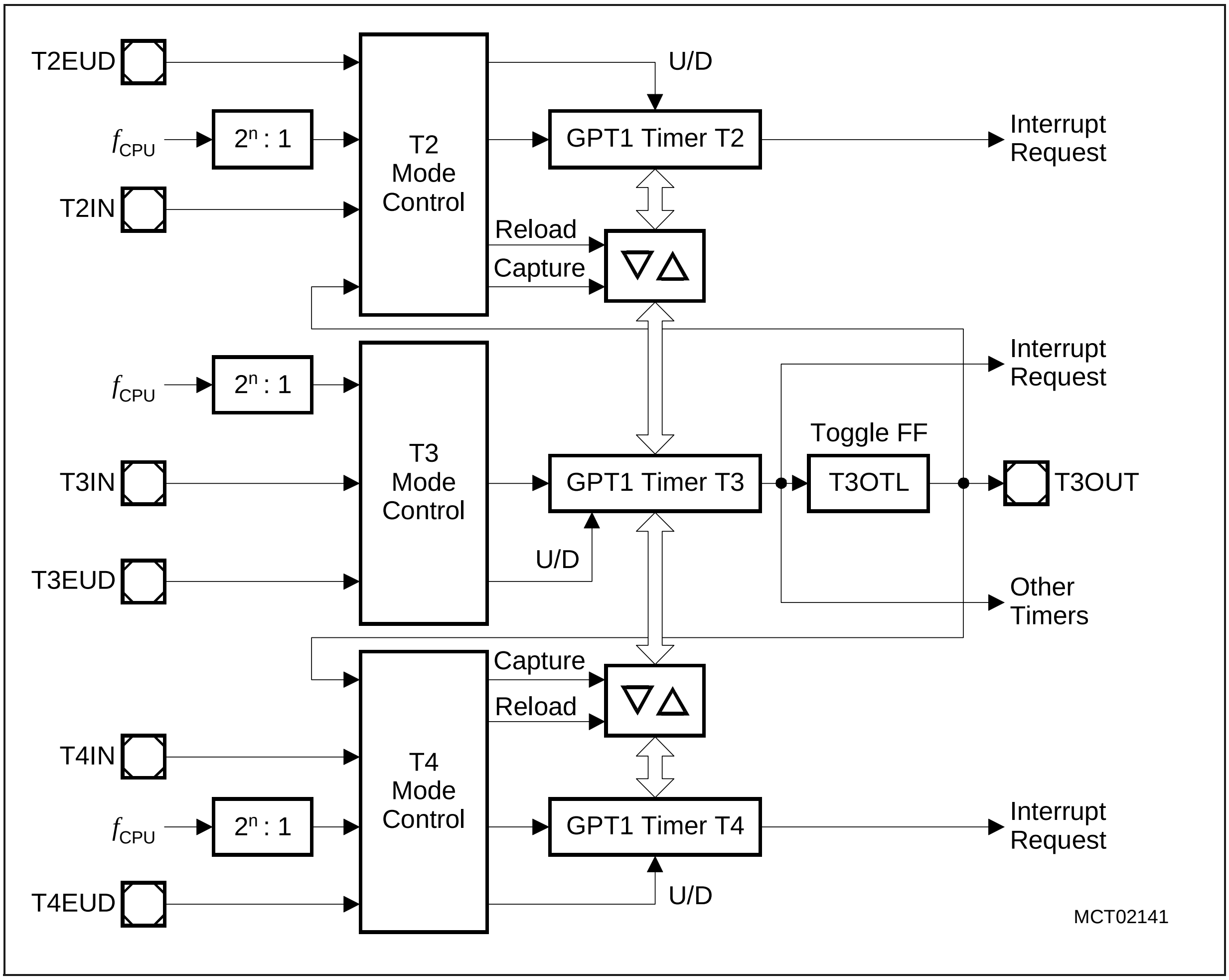


Figure 10-2 GPT1 Block Diagram

10.1.1 GPT1 Core Timer T3

The core timer T3 is configured and controlled via its bitaddressable control register T3CON.

T3CON
Timer 3 Control Register **SFR ($FF42_H$/$A1_H$)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**-**	**-**	**-**	**-**	**T3 OTL**	**-**	**T3 UDE**	**T3 UD**	**T3R**	**T3M**			**T3I**		
-	-	-	-	-	rwh	-	rw	rw	rw	rw			rw		

Bit	Function
T3I	**Timer 3 Input Selection** Depends on the operating mode, see respective sections.
T3M	**Timer 3 Mode Control** (Basic Operating Mode) 000: Timer Mode 001: Counter Mode 010: Gated Timer with Gate active low 011: Gated Timer with Gate active high 100: *Reserved*. Do not use this combination. 101: *Reserved*. Do not use this combination. 110: Incremental Interface Mode 111: *Reserved*. Do not use this combination.
T3R	**Timer 3 Run Bit** 0: Timer / Counter 3 stops 1: Timer / Counter 3 runs
T3UD	**Timer 3 Up / Down Control** *)
T3UDE	**Timer 3 External Up/Down Enable** *)
T3OTL	**Timer 3 Output Toggle Latch** Toggles on each overflow / underflow of T3. Can be set or reset by software.

*) For the effects of bits T3UD and T3UDE refer to the direction table below.

Timer 3 Run Bit

The timer can be started or stopped by software through bit T3R (Timer T3 Run Bit). If T3R=‘0’, the timer stops. Setting T3R to ‘1’ will start the timer.
In gated timer mode, the timer will only run if T3R=‘1’ and the gate is active (high or low, as programmed).

Count Direction Control

The count direction of the core timer can be controlled either by software or by the external input pin T3EUD (Timer T3 External Up/Down Control Input), which is an alternate port input function. These options are selected by bits T3UD and T3UDE in control register T3CON. When the up/down control is done by software (bit T3UDE=‘0’), the count direction can be altered by setting or clearing bit T3UD. When T3UDE=‘1’, pin T3EUD is selected to be the controlling source of the count direction. However, bit T3UD can still be used to reverse the actual count direction, as shown in the table below. If T3UD=‘0’ and pin T3EUD shows a low level, the timer is counting up. With a high level at T3EUD the timer is counting down. If T3UD=‘1’, a high level at pin T3EUD specifies counting up, and a low level specifies counting down. The count direction can be changed regardless of whether the timer is running or not.

When pin T3EUD is used as external count direction control input, it must be configured as input, i.e. its corresponding direction control bit must be set to ‘0’.

Table 10-1 GPT1 Core Timer T3 Count Direction Control

Pin TxEUD	Bit TxUDE	Bit TxUD	Count Direction
X	0	0	Count Up
X	0	1	Count Down
0	1	0	Count Up
1	1	0	Count Down
0	1	1	Count Down
1	1	1	Count Up

Note: The direction control works the same for core timer T3 and for auxiliary timers T2 and T4. Therefore the pins and bits are named Tx...

Timer 3 Output Toggle Latch

An overflow or underflow of timer T3 will clock the toggle bit T3OTL in control register T3CON. T3OTL can also be set or reset by software.

In addition, T3OTL can be used in conjunction with the timer over/underflows as an input for the counter function or as a trigger source for the reload function of the auxiliary timers T2 and T4. An internal connection is provided for this option.

Timer 3 in Timer Mode

Timer mode for the core timer T3 is selected by setting bit field T3M in register T3CON to ‘000_B’. In this mode, T3 is clocked with the internal system clock (CPU clock) divided by a programmable prescaler, which is selected by bit field T3I. The input frequency f_{T3} for timer T3 and its resolution r_{T3} are scaled linearly with lower clock frequencies f_{CPU}, as can be seen from the following formula:

$$f_{T3} = \frac{f_{CPU}}{8 * 2^{<T3I>}} \qquad r_{T3}\,[\mu s] = \frac{8 * 2^{<T3I>}}{f_{CPU}\,[MHz]}$$

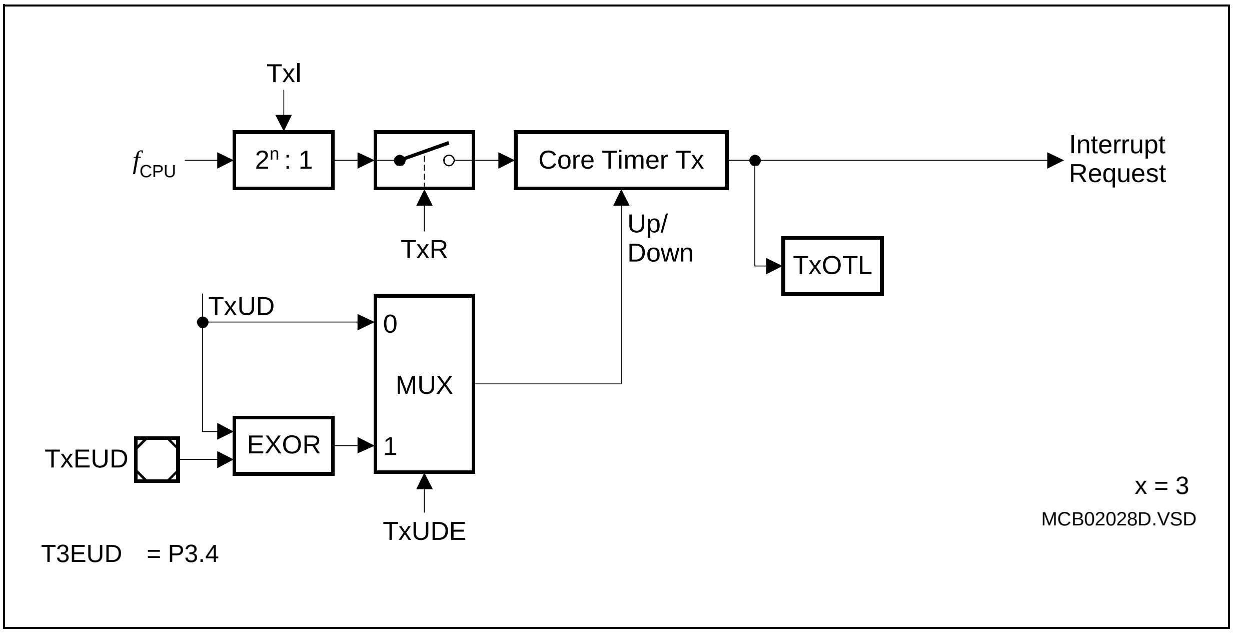


Figure 10-3 Block Diagram of Core Timer T3 in Timer Mode

The timer input frequencies, resolution and periods which result from the selected prescaler option are listed in the table below. This table also applies to the Gated Timer Mode of T3 and to the auxiliary timers T2 and T4 in timer and gated timer mode. Note that some numbers may be rounded to 3 significant digits.

Table 10-2 GPT1 Timer Input Frequencies, Resolution and Periods

f_{CPU} = 20MHz	Timer Input Selection T2I / T3I / T4I							
	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B
Prescaler factor	8	16	32	64	128	256	512	1024
Input Frequency	2.5 MHz	1.25 MHz	625 kHz	312.5 kHz	156.25 kHz	78.125 kHz	39.06 kHz	19.53 kHz
Resolution	400 ns	800 ns	1.6 µs	3.2 µs	6.4 µs	12.8 µs	25.6 µs	51.2 µs
Period	26 ms	52.5 ms	105 ms	210 ms	420 ms	840 ms	1.68 s	3.36 s

Timer 3 in Gated Timer Mode

Gated timer mode for the core timer T3 is selected by setting bit field T3M in register T3CON to ‘010_B’ or ‘011_B’. Bit T3M.0 (T3CON.3) selects the active level of the gate input. In gated timer mode the same options for the input frequency as for the timer mode are available. However, the input clock to the timer in this mode is gated by the external input pin T3IN (Timer T3 External Input).
To enable this operation pin T3IN must be configured as input, i.e. the corresponding direction control bit must contain ‘0’.

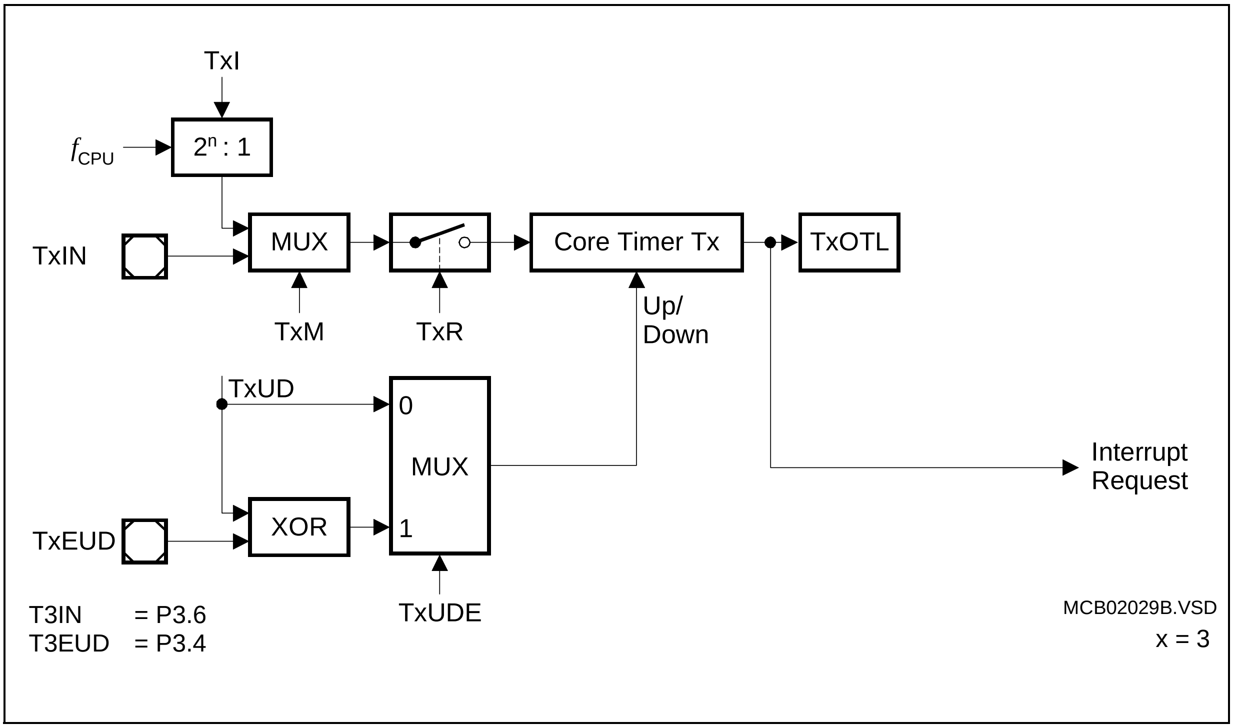


Figure 10-4 Block Diagram of Core Timer T3 in Gated Timer Mode

If T3M.0=‘0’, the timer is enabled when T3IN shows a low level. A high level at this pin stops the timer. If T3M.0=‘1’, pin T3IN must have a high level in order to enable the timer. In addition, the timer can be turned on or off by software using bit T3R. The timer will only run, if T3R=‘1’ and the gate is active. It will stop, if either T3R=‘0’ or the gate is inactive.

Note: A transition of the gate signal at pin T3IN does not cause an interrupt request.

Timer 3 in Counter Mode

Counter mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '001$_B$'. In counter mode timer T3 is clocked by a transition at the external input pin T3IN. The event causing an increment or decrement of the timer can be a positive, a negative, or both a positive and a negative transition at this pin. Bit field T3I in control register T3CON selects the triggering transition (see table below).

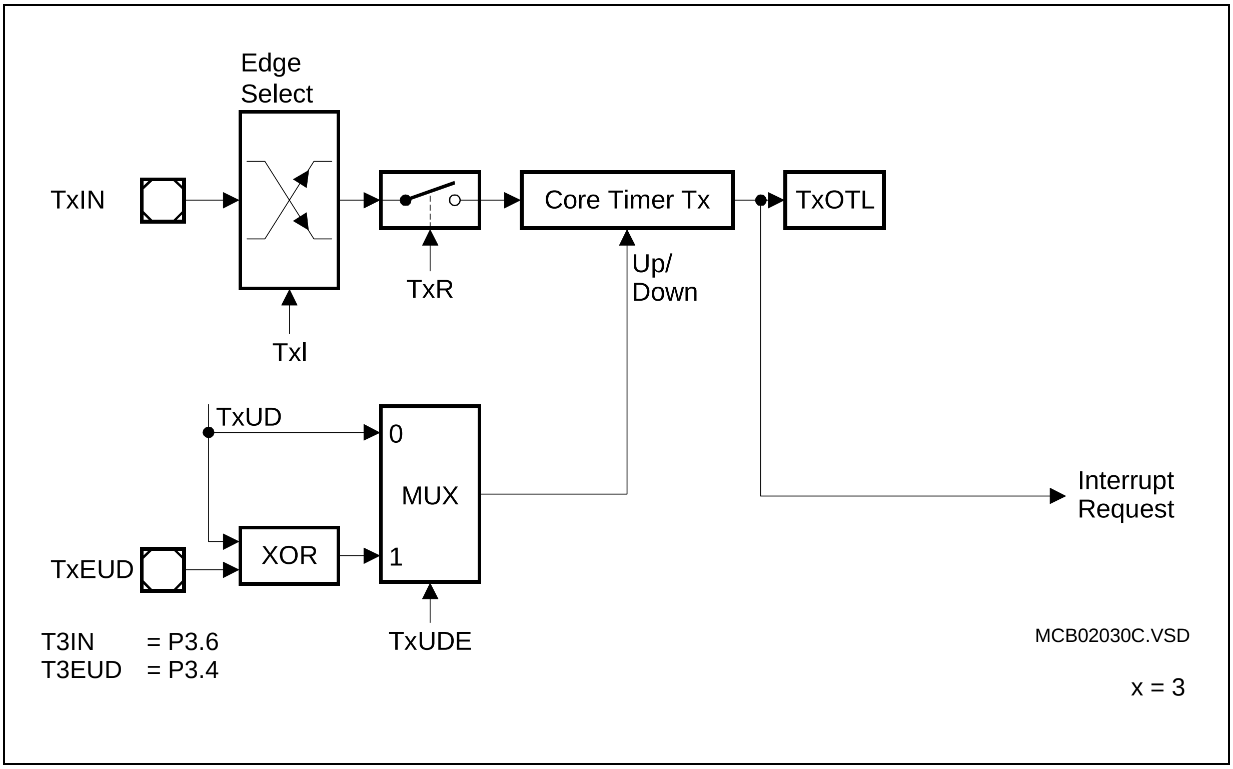


Figure 10-5 Block Diagram of Core Timer T3 in Counter Mode

Table 10-3 GPT1 Core Timer T3 (Counter Mode) Input Edge Selection

T3I	Triggering Edge for Counter Increment / Decrement
0 0 0	None. Counter T3 is disabled
0 0 1	Positive transition (rising edge) on T3IN
0 1 0	Negative transition (falling edge) on T3IN
0 1 1	Any transition (rising or falling edge) on T3IN
1 X X	Reserved. Do not use this combination

For counter operation, pin T3IN must be configured as input, i.e. the respective direction control bit DPx.y must be '0'. The maximum input frequency which is allowed in counter mode is f_{CPU}/16. To ensure that a transition of the count input signal which is applied to T3IN is correctly recognized, its level should be held high or low for at least 8 f_{CPU} cycles before it changes.

Timer 3 in Incremental Interface Mode

Incremental Interface mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '110_B'. In incremental interface mode the two inputs associated with timer T3 (T3IN, T3EUD) are used to interface to an incremental encoder. T3 is clocked by each transition on one or both of the external input pins which gives 2-fold or 4-fold resolution of the encoder input.

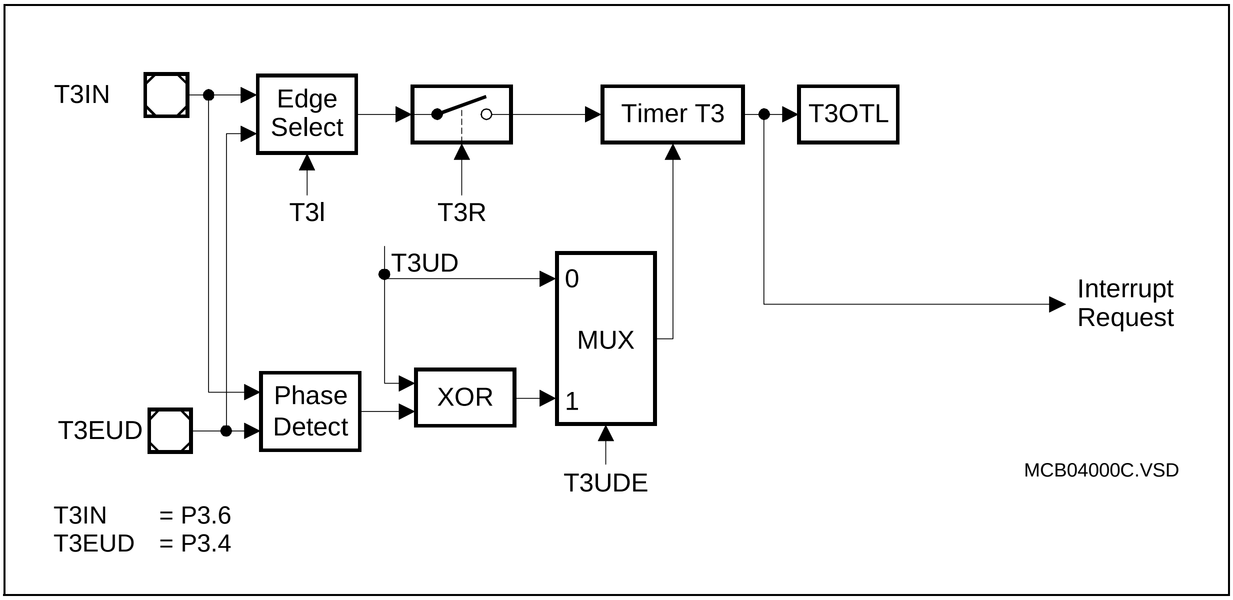


Figure 10-6 Block Diagram of Core Timer T3 in Incremental Interface Mode

Bitfield T3I in control register T3CON selects the triggering transitions (see table below). In this mode the sequence of the transitions of the two input signals is evaluated and generates count pulses as well as the direction signal. So T3 is modified automatically according to the speed and the direction of the incremental encoder and its contents therefore always represent the encoder's current position.

Table 10-4 GPT1 Core Timer T3 Input Edge Selection in Incremental Interface Mode

T3I	Triggering Edge for Counter Increment / Decrement
0 0 0	None. Counter T3 stops.
0 0 1	Any transition (rising or falling edge) on T3IN.
0 1 0	Any transition (rising or falling edge) on T3EUD.
0 1 1	Any transition (rising or falling edge) on any T3 input (T3IN or T3EUD).
1 X X	Reserved. Do not use this combination

The incremental encoder can be connected directly to the C164 without external interface logic. In a standard system, however, comparators will be employed to convert the encoder's differential outputs (e.g. A, $\overline{A}$) to digital signals (e.g. A). This greatly increases noise immunity.

*Note: The third encoder output Top0, which indicates the mechanical zero position, may be connected to an external interrupt input and trigger a reset of timer T3 (*e.g. *via PEC transfer from ZEROS).*

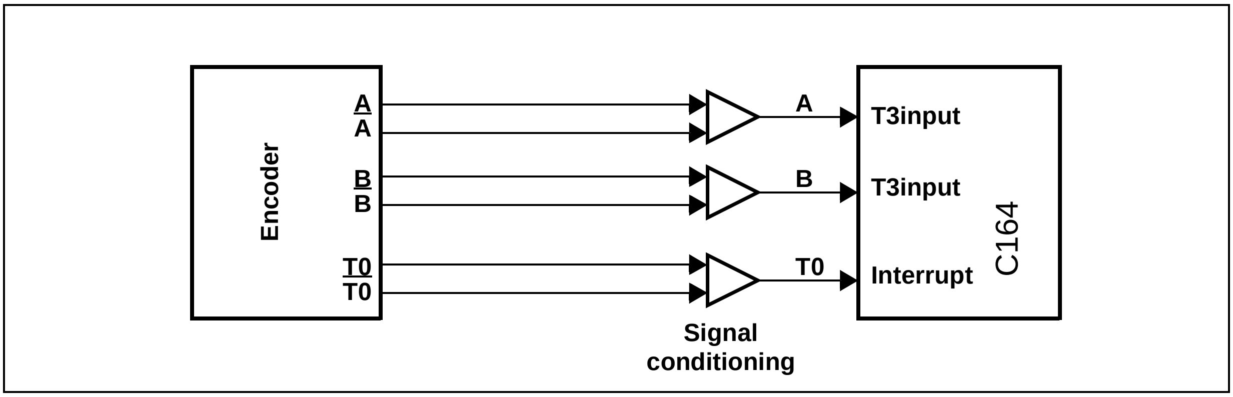


Figure 10-7 Connection of the Encoder to the C164

For incremental interface operation the following conditions must be met:

- Bitfield T3M must be '110$_B$'.
- Both pins T3IN and T3EUD must be configured as input, i.e. the respective direction control bits must be '0'.
- Bit T3UDE must be '1' to enable automatic direction control.

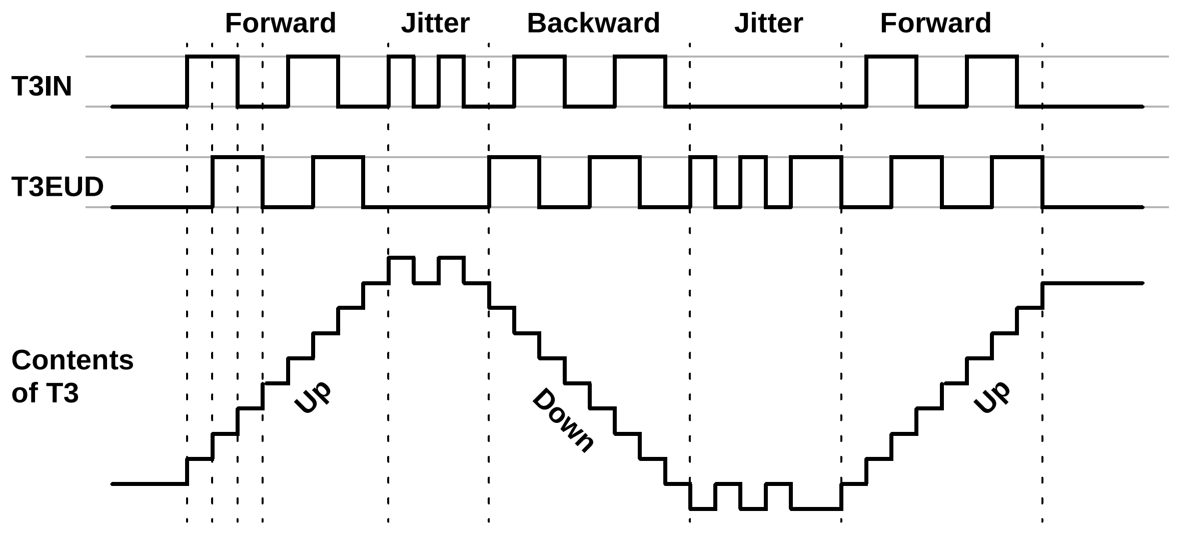
The maximum input frequency which is allowed in incremental interface mode is f_{CPU}/16. To ensure that a transition of any input signal is correctly recognized, its level should be held high or low for at least 8 f_{CPU} cycles before it changes.

In Incremental Interface Mode the count direction is automatically derived from the sequence in which the input signals change, which corresponds to the rotation direction of the connected sensor. The table below summarizes the possible combinations.

Table 10-5 GPT1 Core Timer T3 Count Direction in Incremental Interface Mode

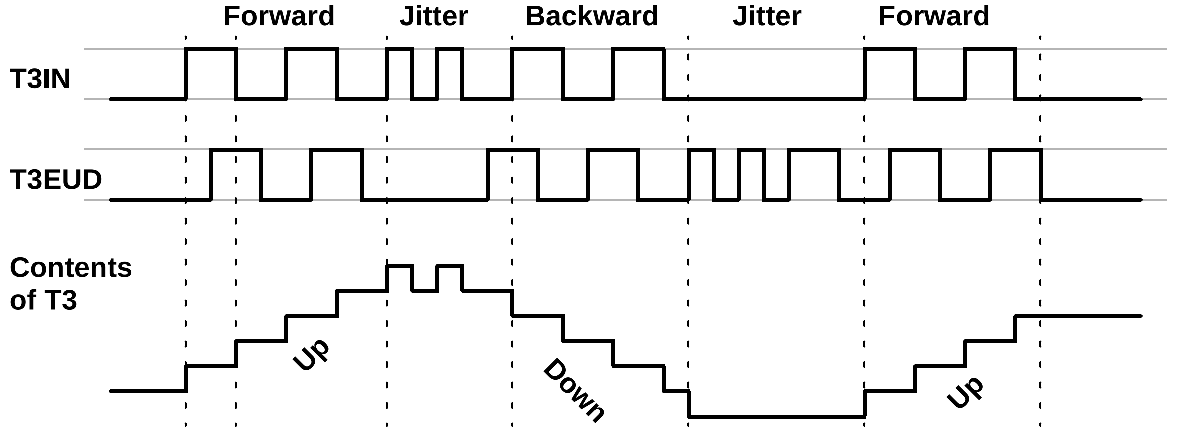
Level on respective other input	T3IN Input		T3EUD Input	
	Rising ↗	Falling ↘	Rising ↗	Falling ↘
High	Down	Up	Up	Down
Low	Up	Down	Down	Up

The figures below give examples of T3's operation, visualizing count signal generation and direction control. It also shows how input jitter is compensated which might occur if the sensor rests near to one of its switching points.



Note: This example shows the timer behaviour assuming that T3 counts upon any transition on any input, i.e. T3I = '011$_B$'.

Figure 10-8 Evaluation of the Incremental Encoder Signals



Note: This example shows the timer behaviour assuming that T3 counts upon any transition on input T3IN, i.e. T3I = '001_B'.

Figure 10-9 Evaluation of the Incremental Encoder Signals

Note: Timer T3 operating in incremental interface mode automatically provides information on the sensor's current position. Dynamic information (speed, acceleration, deceleration) may be obtained by measuring the incoming signal periods.

10.1.2 GPT1 Auxiliary Timers T2 and T4

Both auxiliary timers T2 and T4 have exactly the same functionality. They can be configured for timer, gated timer, counter, or incremental interface mode with the same options for the timer frequencies and the count signal as the core timer T3. In addition to these 4 counting modes, the auxiliary timers can be concatenated with the core timer, or they may be used as reload or capture registers in conjunction with the core timer.

The individual configuration for timers T2 and T4 is determined by their bitaddressable control registers T2CON and T4CON, which are both organized identically. Note that functions which are present in all 3 timers of block GPT1 are controlled in the same bit positions and in the same manner in each of the specific control registers.

T2CON
Timer 2 Control Register **SFR ($FF40_H$/$A0_H$)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	T2 UDE	T2 UD	T2R	T2M			T2I		
-	-	-	-	-	-	-	rw	rw	rw	rw			rw		

T4CON
Timer 4 Control Register **SFR ($FF44_H$/$A2_H$)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	T4 UDE	T4 UD	T4R	T4M			T4I		
-	-	-	-	-	-	-	rw	rw	rw	rw			rw		

Bit	Function
TxI	**Timer x Input Selection** Depends on the Operating Mode, see respective sections.
TxM	**Timer x Mode Control** (Basic Operating Mode) 000: Timer Mode 001: Counter Mode 010: Gated Timer with Gate active low 011: Gated Timer with Gate active high 100: Reload Mode 101: Capture Mode 110: Incremental Interface Mode 111: *Reserved*. Do not use this combination.
TxR	**Timer x Run Bit** 0: Timer / Counter x stops 1: Timer / Counter x runs

Bit	Function
TxUD	**Timer x Up / Down Control** *)
TxUDE	**Timer x External Up/Down Enable** *)

*) For the effects of bits TxUD and TxUDE refer to the direction table (see T3 section).

Note: The auxiliary timers have no output toggle latch and no alternate output function.

Count Direction Control for Auxiliary Timers

The count direction of the auxiliary timers can be controlled in the same way as for the core timer T3. The description and the table apply accordingly.

Timers T2 and T4 in Timer Mode or Gated Timer Mode

When the auxiliary timers T2 and T4 are programmed to timer mode or gated timer mode, their operation is the same as described for the core timer T3. The descriptions, figures and tables apply accordingly with one exception:

- There is no output toggle latch for T2 and T4.

Timers T2 and T4 in Incremental Interface Mode

When the auxiliary timers T2 and T4 are programmed to incremental interface mode, their operation is the same as described for the core timer T3. The descriptions, figures and tables apply accordingly.

Timers T2 and T4 in Counter Mode

Counter mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective register TxCON to ‘001_B’. In counter mode timers T2 and T4 can be clocked either by a transition at the respective external input pin TxIN, or by a transition of timer T3’s output toggle latch T3OTL.

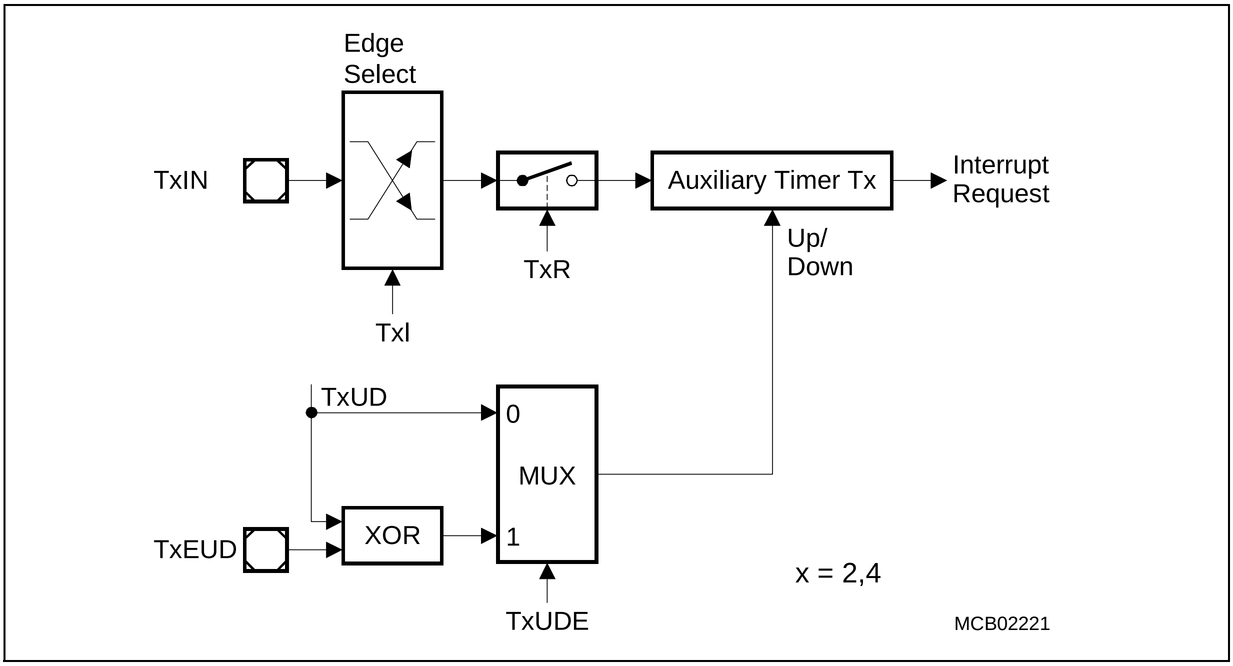


Figure 10-10 Block Diagram of an Auxiliary Timer in Counter Mode

The event causing an increment or decrement of a timer can be a positive, a negative, or both a positive and a negative transition at either the respective input pin, or at the toggle latch T3OTL.

Bit field TxI in the respective control register TxCON selects the triggering transition (see table below).

Table 10-6 GPT1 Auxiliary Timer Input Edge Selection in Counter Mode

T2I / T4I	Triggering Edge for Counter Increment / Decrement
X 0 0	None. Counter Tx is disabled
0 0 1	Positive transition (rising edge) on TxIN
0 1 0	Negative transition (falling edge) on TxIN
0 1 1	Any transition (rising or falling edge) on TxIN
1 0 1	Positive transition (rising edge) of output toggle latch T3OTL
1 1 0	Negative transition (falling edge) of output toggle latch T3OTL
1 1 1	Any transition (rising or falling edge) of output toggle latch T3OTL

Note: Only state transitions of T3OTL which are caused by the overflows/underflows of T3 will trigger the counter function of T2/T4. Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.

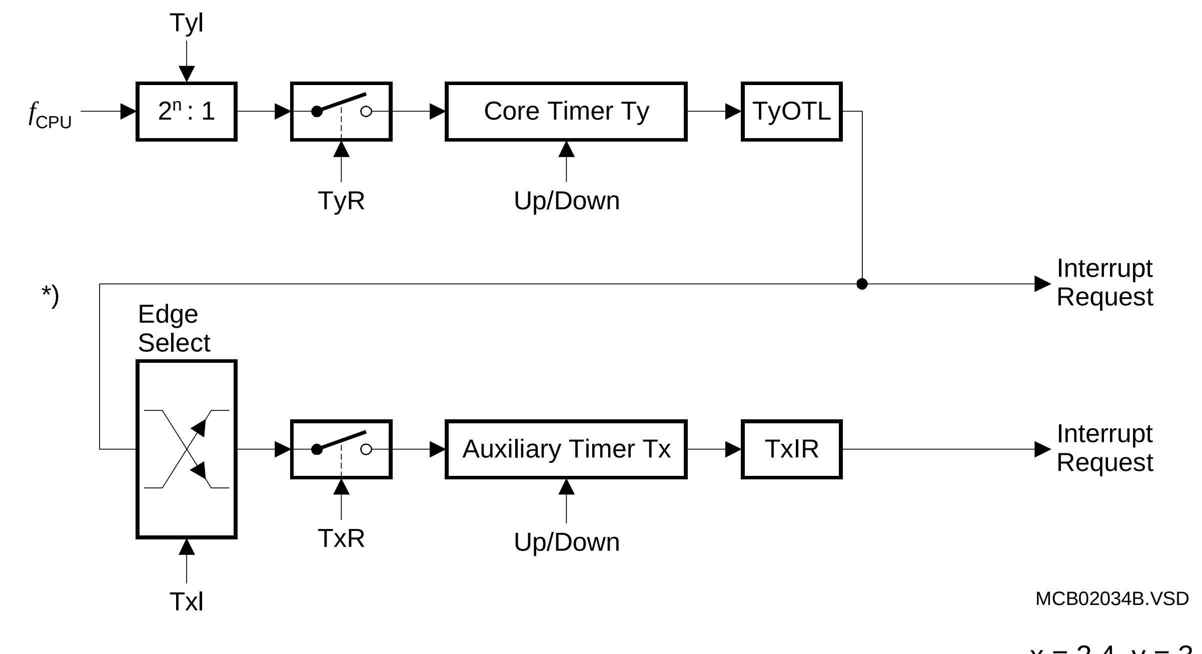
For counter operation, pin TxIN must be configured as input, i.e. the respective direction control bit must be ‘0’. The maximum input frequency which is allowed in counter mode is f_{CPU}/16. To ensure that a transition of the count input signal which is applied to TxIN is correctly recognized, its level should be held for at least 8 f_{CPU} cycles before it changes.

Timer Concatenation

Using the toggle bit T3OTL as a clock source for an auxiliary timer in counter mode concatenates the core timer T3 with the respective auxiliary timer. Depending on which transition of T3OTL is selected to clock the auxiliary timer, this concatenation forms a 32-bit or a 33-bit timer/counter.

- **32-bit Timer/Counter**: If both a positive and a negative transition of T3OTL is used to clock the auxiliary timer, this timer is clocked on every overflow/underflow of the core timer T3. Thus, the two timers form a 32-bit timer.
- **33-bit Timer/Counter**: If either a positive or a negative transition of T3OTL is selected to clock the auxiliary timer, this timer is clocked on every second overflow/underflow of the core timer T3. This configuration forms a 33-bit timer (16-bit core timer+T3OTL+16-bit auxiliary timer).

The count directions of the two concatenated timers are not required to be the same. This offers a wide variety of different configurations.
T3 can operate in timer, gated timer or counter mode in this case.



x = 2,4 y = 3

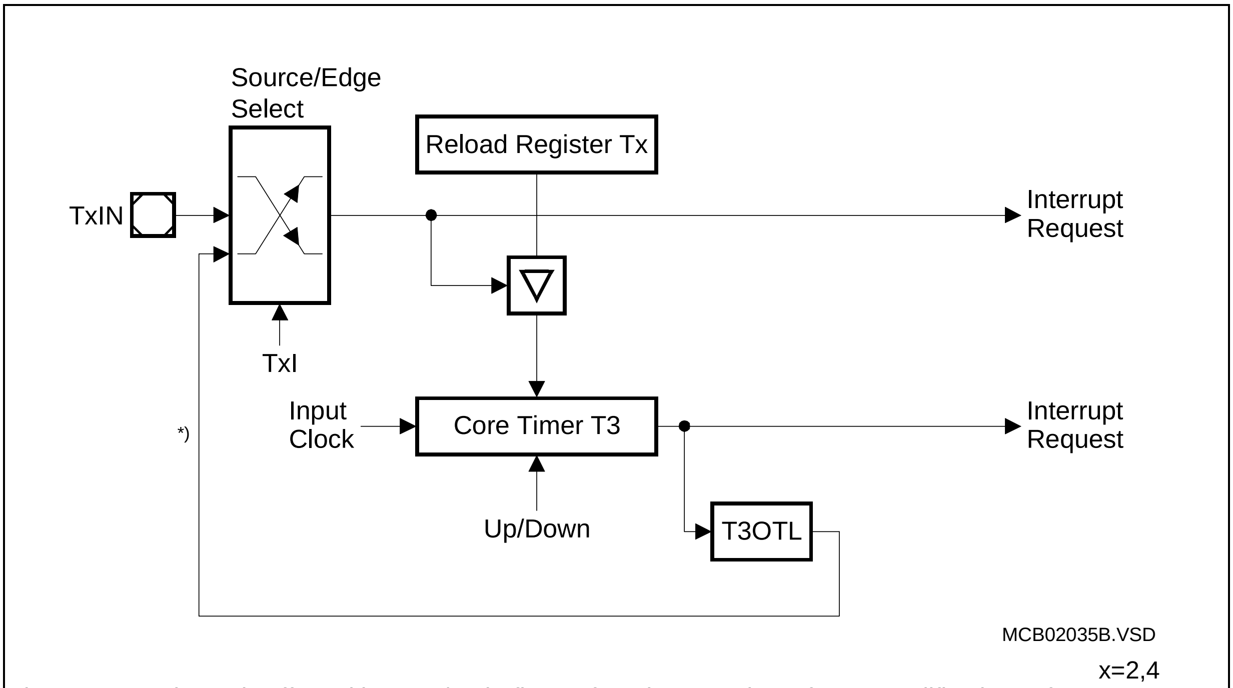
*) Note: Line only affected by over/underflows of T3, but NOT by software modifications of T3OTL.

Figure 10-11 Concatenation of Core Timer T3 and an Auxiliary Timer

Auxiliary Timer in Reload Mode

Reload mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective register TxCON to '100_B'. In reload mode the core timer T3 is reloaded with the contents of an auxiliary timer register, triggered by one of two different signals. The trigger signal is selected the same way as the clock source for counter mode (see table above), i.e. a transition of the auxiliary timer's input or the output toggle latch T3OTL may trigger the reload.

Note: When programmed for reload mode, the respective auxiliary timer (T2 or T4) stops independent of its run flag T2R or T4R.



*) Note: Line only affected by over/underflows of T3, but NOT by software modifications of T3OTL.

Figure 10-12 GPT1 Auxiliary Timer in Reload Mode

Upon a trigger signal T3 is loaded with the contents of the respective timer register (T2 or T4) and the interrupt request flag (T2IR or T4IR) is set.

Note: When a T3OTL transition is selected for the trigger signal, also the interrupt request flag T3IR will be set upon a trigger, indicating T3's overflow or underflow. Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.

The reload mode triggered by T3OTL can be used in a number of different configurations. Depending on the selected active transition the following functions can be performed:

- If both a positive and a negative transition of T3OTL is selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer each time it overflows or underflows. This is the standard reload mode (reload on overflow/ underflow).
- If either a positive or a negative transition of T3OTL is selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer on every second overflow or underflow.
- Using this “single-transition” mode for both auxiliary timers allows to perform very flexible pulse width modulation (PWM). One of the auxiliary timers is programmed to reload the core timer on a positive transition of T3OTL, the other is programmed for a reload on a negative transition of T3OTL. With this combination the core timer is alternately reloaded from the two auxiliary timers.

Note: Although it is possible, it should be avoided to select the same reload trigger event for both auxiliary timers. In this case both reload registers would try to load the core timer at the same time. If this combination is selected, T2 disregarded and the contents of T4 is reloaded.

Auxiliary Timer in Capture Mode

Capture mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective register TxCON to '101_B'. In capture mode the contents of the core timer are latched into an auxiliary timer register in response to a signal transition at the respective auxiliary timer's external input pin TxIN. The capture trigger signal can be a positive, a negative, or both a positive and a negative transition.

The two least significant bits of bit field TxI are used to select the active transition (see table in the counter mode section), while the most significant bit TxI.2 is irrelevant for capture mode. It is recommended to keep this bit cleared (TxI.2 = '0').

Note: When programmed for capture mode, the respective auxiliary timer (T2 or T4) stops independent of its run flag T2R or T4R.

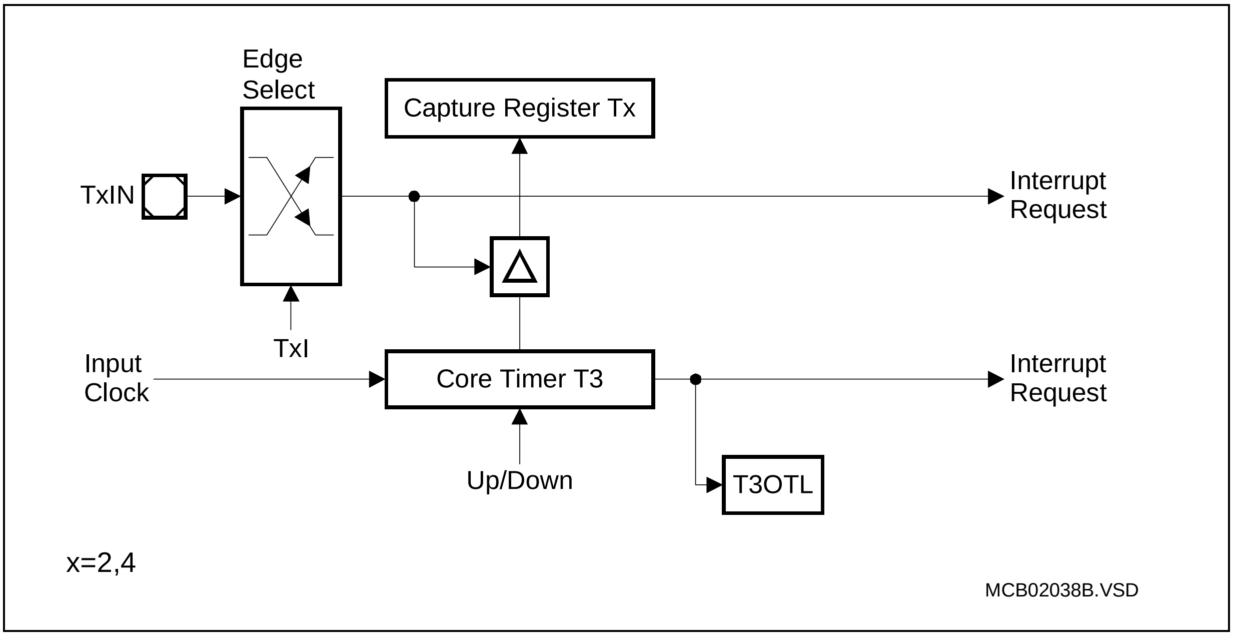


Figure 10-13 GPT1 Auxiliary Timer in Capture Mode

Upon a trigger (selected transition) at the corresponding input pin TxIN the contents of the core timer are loaded into the auxiliary timer register and the associated interrupt request flag TxIR will be set.

Note: The direction control bits for T2IN and T4IN must be set to '0', and the level of the capture trigger signal should be held high or low for at least 8 f_{CPU} cycles before it changes to ensure correct edge detection.

10.1.3 Interrupt Control for GPT1 Timers

When a timer overflows from FFFF$_H$ to 0000$_H$ (when counting up), or when it underflows from 0000$_H$ to FFFF$_H$ (when counting down), its interrupt request flag (T2IR, T3IR or T4IR) in register TxIC will be set. This will cause an interrupt to the respective timer interrupt vector (T2INT, T3INT or T4INT) or trigger a PEC service, if the respective interrupt enable bit (T2IE, T3IE or T4IE in register TxIC) is set. There is an interrupt control register for each of the three timers.

T2IC
Timer 2 Intr. Ctrl. Reg. SFR (FF60$_H$/B0$_H$) Reset value: - - 00$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								**T2IR**	**T2IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

T3IC
Timer 3 Intr. Ctrl. Reg. SFR (FF62$_H$/B1$_H$) Reset value: - - 00$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								**T3IR**	**T3IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

T4IC
Timer 4 Intr. Ctrl. Reg. SFR (FF64$_H$/B2$_H$) Reset value: - - 00$_H$

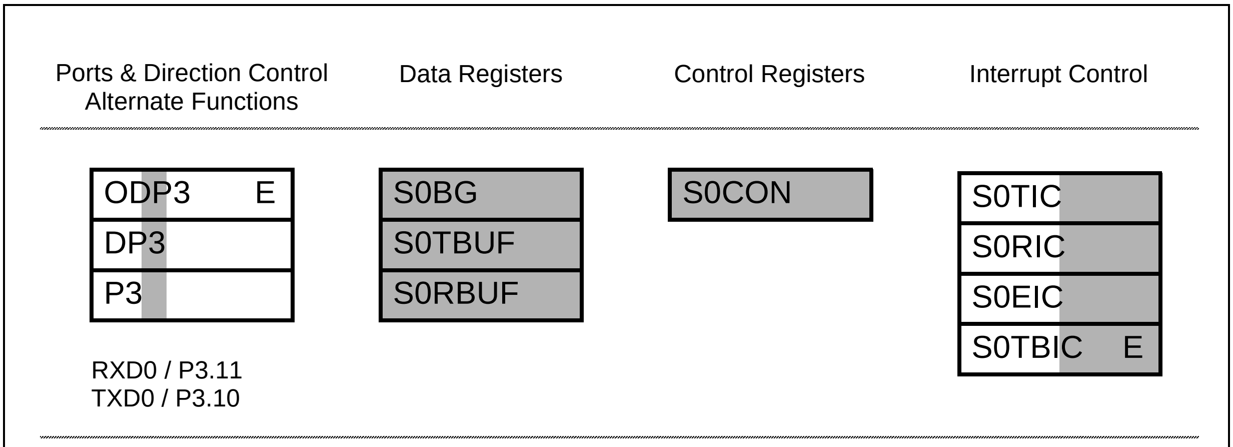
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								**T4IR**	**T4IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

11 The Asynchronous/Synchronous Serial Interface

The Asynchronous/Synchronous Serial Interface ASC0 provides serial communication between the C164 and other microcontrollers, microprocessors or external peripherals.

The ASC0 supports full-duplex asynchronous communication up to 780 KBaud and half-duplex synchronous communication up to 3.1 MBaud (@ 25 MHz CPU clock). In synchronous mode, data are transmitted or received synchronous to a shift clock which is generated by the C164. In asynchronous mode, 8- or 9-bit data transfer, parity generation, and the number of stop bits can be selected. Parity, framing, and overrun error detection is provided to increase the reliability of data transfers. Transmission and reception of data is double-buffered. For multiprocessor communication, a mechanism to distinguish address from data bytes is included. Testing is supported by a loop-back option. A 13-bit baud rate generator provides the ASC0 with a separate serial clock signal.



ODP3	Port 3 Open Drain Control Register	P3	Port 3 Data Register
DP3	Port 3 Direction Control Register	S0CON	ASC0 Control Register
S0BG	ASC0 Baud Rate Generator/Reload Reg.	S0RBUF	ASC0 Receive Buffer Register (read only)
S0TBUF	ASC0 Transmit Buffer Register	S0RIC	ASC0 Receive Interrupt Control Register
S0TIC	ASC0 Transmit Interrupt Control Register	S0EIC	ASC0 Error Interrupt Control Register
S0TBIC	ASC0 Transmit Buffer Interrupt Ctrl. Reg.		

Figure 11-1 SFRs and Port Pins associated with ASC0

The operating mode of the serial channel ASC0 is controlled by its bitaddressable control register S0CON. This register contains control bits for mode and error check selection, and status flags for error identification.

S0CON
ASC0 Control Register **SFR ($FFB0_H$/$D8_H$)** **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S0R	**S0 LB**	**S0 BRS**	**S0 ODD**	**-**	**S0 OE**	**S0 FE**	**S0 PE**	**S0 OEN**	**S0 FEN**	**S0 PEN**	**S0 REN**	**S0 STP**		**S0M**	
rw	rw	rw	rw	-	rwh	rwh	rwh	rw	rw	rw	rwh	rw		rw	

Bit	Function
S0M	**ASC0 Mode Control** 000: 8-bit data synchronous operation 001: 8-bit data async. operation 010: Reserved. Do not use this combination! 011: 7-bit data + parity async. operation 100: 9-bit data async. operation 101: 8-bit data + wake up bit async. operation 110: Reserved. Do not use this combination! 111: 8-bit data + parity async. operation
S0STP	**Number of Stop Bits Selection** async. operation 0: One stop bit 1: Two stop bits
S0REN	**Receiver Enable Bit** 0: Receiver disabled 1: Receiver enabled (Reset by hardware after reception of byte in synchronous mode)
S0PEN	**Parity Check Enable Bit** async. operation 0: Ignore parity 1: Check parity
S0FEN	**Framing Check Enable Bit** async. operation 0: Ignore framing errors 1: Check framing errors
S0OEN	**Overrun Check Enable Bit** 0: Ignore overrun errors 1: Check overrun errors
S0PE	**Parity Error Flag** Set by hardware on a parity error (S0PEN='1'). Must be reset by software.

Bit	Function
S0FE	**Framing Error Flag** Set by hardware on a framing error (S0FEN='1'). Must be reset by software.
S0OE	**Overrun Error Flag** Set by hardware on an overrun error (S0OEN='1'). Must be reset by software.
S0ODD	**Parity Selection Bit** 0: Even parity (parity bit set on odd number of '1's in data) 1: Odd parity (parity bit set on even number of '1's in data)
S0BRS	**Baudrate Selection Bit** 0: Divide clock by reload-value + constant (depending on mode) 1: Additionally reduce serial clock to 2/3rd
S0LB	**LoopBack Mode Enable Bit** 0: Standard transmit/receive mode 1: Loopback mode enabled
S0R	**Baudrate Generator Run Bit** 0: Baudrate generator disabled (ASC0 inactive) 1: Baudrate generator enabled

A transmission is started by writing to the Transmit Buffer register S0TBUF (via an instruction or a PEC data transfer). Only the number of data bits which is determined by the selected operating mode will actually be transmitted, i.e. bits written to positions 9 through 15 of register S0TBUF are always insignificant. After a transmission has been completed, the transmit buffer register is cleared to 0000_H.

Data transmission is double-buffered, so a new character may be written to the transmit buffer register, before the transmission of the previous character is complete. This allows the transmission of characters back-to-back without gaps.

Data reception is enabled by the Receiver Enable Bit S0REN. After reception of a character has been completed, the received data and, if provided by the selected operating mode, the received parity bit can be read from the (read-only) Receive Buffer register S0RBUF. Bits in the upper half of S0RBUF which are not valid in the selected operating mode will be read as zeros.

Data reception is double-buffered, so that reception of a second character may already begin before the previously received character has been read out of the receive buffer register. In all modes, receive buffer overrun error detection can be selected through bit S0OEN. When enabled, the overrun error status flag S0OE and the error interrupt request flag S0EIR will be set when the receive buffer register has not been read by the time reception of a second character is complete. The previously received character in the receive buffer is overwritten.

The Loop-Back option (selected by bit S0LB) allows the data currently being transmitted to be received simultaneously in the receive buffer. This may be used to test serial communication routines at an early stage without having to provide an external network. In loop-back mode the alternate input/output functions of the Port 3 pins are not necessary.

Note: Serial data transmission or reception is only possible when the Baud Rate Generator Run Bit S0R is set to '1'. Otherwise the serial interface is idle.
Do not program the mode control field S0M in register S0CON to one of the reserved combinations to avoid unpredictable behaviour of the serial interface.

11.1 Asynchronous Operation

Asynchronous mode supports full-duplex communication, where both transmitter and receiver use the same data frame format and the same baud rate. Data is transmitted on pin TXD0 and received on pin RXD0. These signals are alternate port functions.

Asynchronous Data Frames

8-bit data frames either consist of 8 data bits D7...D0 (S0M='001_B'), or of 7 data bits D6...D0 plus an automatically generated parity bit (S0M='011_B'). Parity may be odd or even, depending on bit S0ODD in register S0CON. An even parity bit will be set, if the modulo-2-sum of the 7 data bits is '1'. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S0PEN (always OFF in 8-bit data mode). The parity error flag S0PE will be set along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit S0RBUF.7.

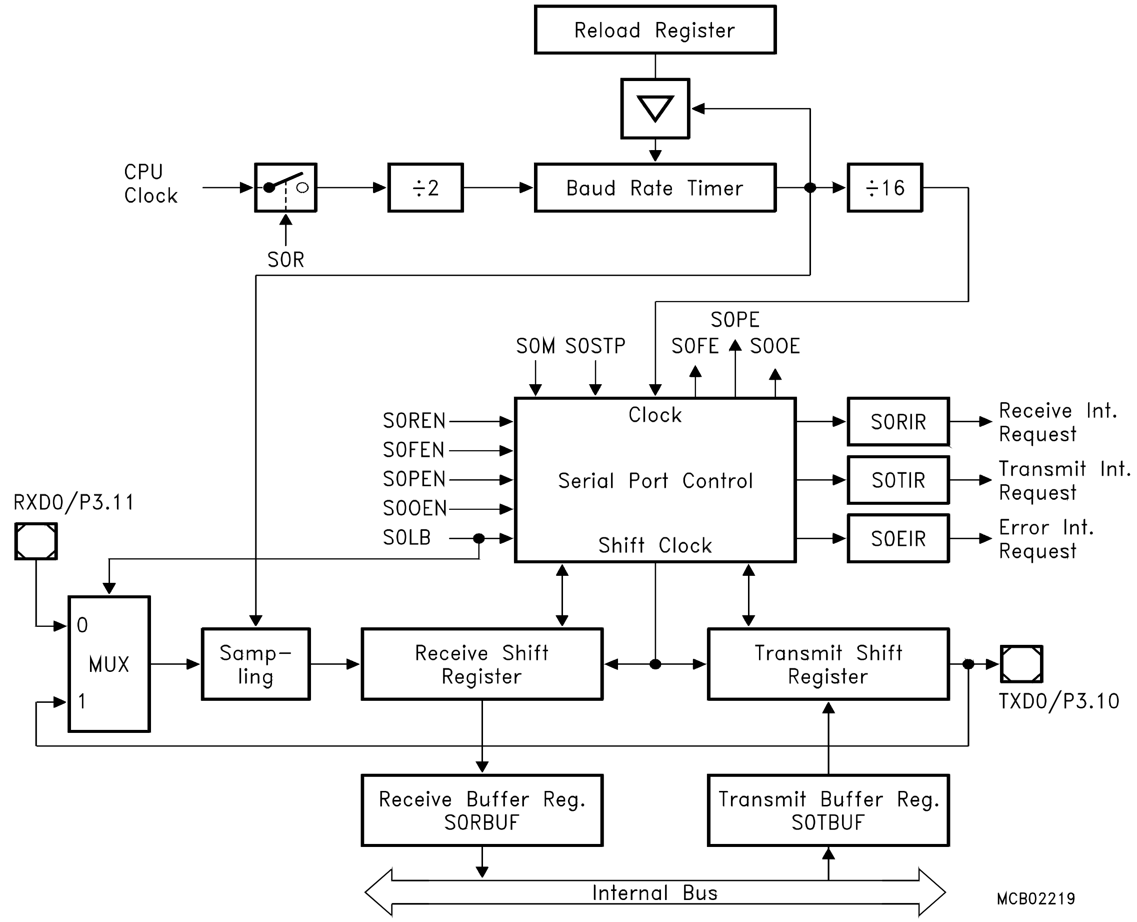


Figure 11-2 Asynchronous Mode of Serial Channel ASC0

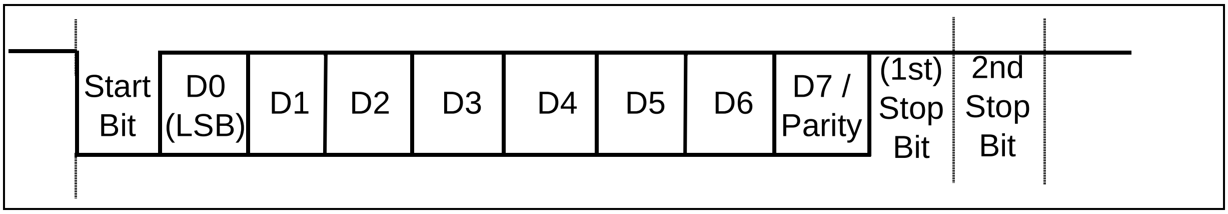


Figure 11-3 Asynchronous 8-bit Data Frames

9-bit data frames either consist of 9 data bits D8...D0 (S0M='100$_B$'), of 8 data bits D7...D0 plus an automatically generated parity bit (S0M='111$_B$') or of 8 data bits D7...D0 plus wake-up bit (S0M='101$_B$'). Parity may be odd or even, depending on bit S0ODD in register S0CON. An even parity bit will be set, if the modulo-2-sum of the 8 data bits is '1'. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S0PEN (always OFF in 9-bit data and wake-up mode). The parity error flag S0PE will be set along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit S0RBUF.8.

In wake-up mode received frames are only transferred to the receive buffer register, if the 9th bit (the wake-up bit) is '1'. If this bit is '0', no receive interrupt request will be activated and no data will be transferred.

This feature may be used to control communication in multi-processor system:
When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the additional 9th bit is a '1' for an address byte and a '0' for a data byte, so no slave will be interrupted by a data 'byte'. An address 'byte' will interrupt all slaves (operating in 8-bit data + wake-up bit mode), so each slave can examine the 8 LSBs of the received character (the address). The addressed slave will switch to 9-bit data mode (e.g. by clearing bit S0M.0), which enables it to also receive the data bytes that will be coming (having the wake-up bit cleared). The slaves that were not being addressed remain in 8-bit data + wake-up bit mode, ignoring the following data bytes.

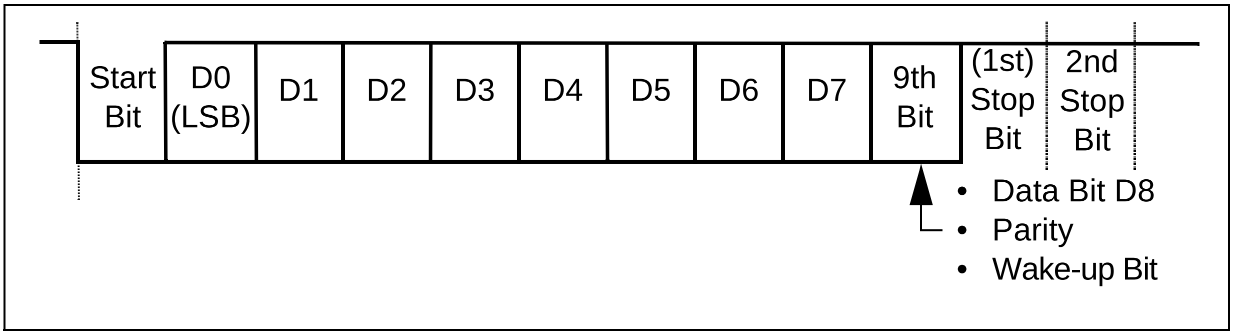


Figure 11-4 Asynchronous 9-bit Data Frames

Asynchronous transmission begins at the next overflow of the divide-by-16 counter (see figure above), provided that S0R is set and data has been loaded into S0TBUF. The transmitted data frame consists of three basic elements:

- the start bit
- the data field (8 or 9 bits, LSB first, including a parity bit, if selected)
- the delimiter (1 or 2 stop bits)

Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into S0TBUF is immediately moved to the transmit shift register thus freeing S0TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request flag S0TBIR being set. S0TBUF may now be loaded with the next data, while transmission of the previous one is still going on.

The transmit interrupt request flag S0TIR will be set before the last bit of a frame is transmitted, i.e. before the first or the second stop bit is shifted out of the transmit shift register.
The transmitter output pin TXD0 must be configured for alternate data output, i.e. the respective port output latch and the direction latch must be ’1’.

Asynchronous reception is initiated by a falling edge (1-to-0 transition) on pin RXD0, provided that bits S0R and S0REN are set. The receive data input pin RXD0 is sampled at 16 times the rate of the selected baud rate. A majority decision of the 7th, 8th and 9th sample determines the effective bit value. This avoids erroneous results that may be caused by noise.

If the detected value is not a '0' when the start bit is sampled, the receive circuit is reset and waits for the next 1-to-0 transition at pin RXD0. If the start bit proves valid, the receive circuit continues sampling and shifts the incoming data frame into the receive shift register.

When the last stop bit has been received, the content of the receive shift register is transferred to the receive data buffer register S0RBUF. Simultaneously, the receive interrupt request flag S0RIR is set after the 9th sample in the last stop bit time slot (as programmed), regardless whether valid stop bits have been received or not. The receive circuit then waits for the next start bit (1-to-0 transition) at the receive data input pin.

The receiver input pin RXD0 must be configured for input, i.e. the respective direction latch must be ’0’.

Asynchronous reception is stopped by clearing bit S0REN. A currently received frame is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Start bits that follow this frame will not be recognized.

Note: In wake-up mode received frames are only transferred to the receive buffer register, if the 9th bit (the wake-up bit) is ‘1’. If this bit is ‘0’, no receive interrupt request will be activated and no data will be transferred.

11.2 Synchronous Operation

Synchronous mode supports half-duplex communication, basically for simple IO expansion via shift registers. Data is transmitted and received via pin RXD0, while pin TXD0 outputs the shift clock. These signals are alternate port functions. Synchronous mode is selected with S0M='000_B'.

8 data bits are transmitted or received synchronous to a shift clock generated by the internal baud rate generator. The shift clock is only active as long as data bits are transmitted or received.

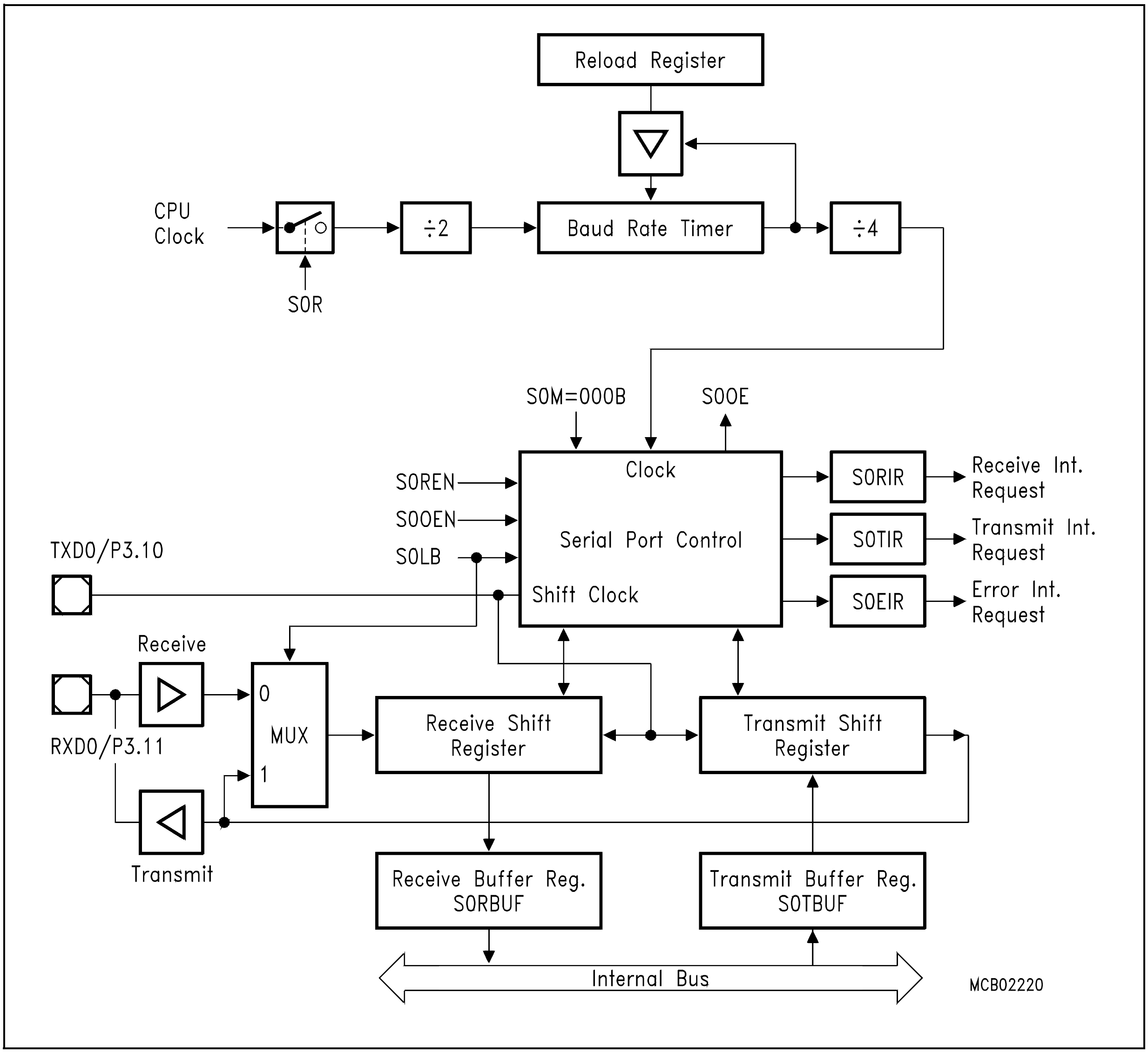


Figure 11-5 Synchronous Mode of Serial Channel ASC0

Synchronous transmission begins within 4 state times after data has been loaded into S0TBUF, provided that S0R is set and S0REN=’0’ (half-duplex, no reception). Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into S0TBUF is immediately moved to the transmit shift register thus freeing S0TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request flag S0TBIR being set. S0TBUF may now be loaded with the next data, while transmission of the previous one is still going on. The data bits are transmitted synchronous with the shift clock. After the bit time for the 8th data bit, both pins TXD0 and RXD0 will go high, the transmit interrupt request flag S0TIR is set, and serial data transmission stops.

Pin TXD0 must be configured for alternate data output, i.e. the respective port output latch and the direction latch must be ’1’, in order to provide the shift clock. Pin RXD0 must also be configured for output (output/direction latch=’1’) during transmission.

Synchronous reception is initiated by setting bit S0REN=’1’. If bit S0R=1, the data applied at pin RXD0 are clocked into the receive shift register synchronous to the clock which is output at pin TXD0. After the 8th bit has been shifted in, the content of the receive shift register is transferred to the receive data buffer S0RBUF, the receive interrupt request flag S0RIR is set, the receiver enable bit S0REN is reset, and serial data reception stops.

Pin TXD0 must be configured for alternate data output, i.e. the respective port output latch and the direction latch must be ’1’, in order to provide the shift clock. Pin RXD0 must be configured as alternate data input, i.e. the respective direction latch must be ’0’.

Synchronous reception is stopped by clearing bit S0REN. A currently received byte is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Writing to the transmit buffer register while a reception is in progress has no effect on reception and will not start a transmission.

If a previously received byte has not been read out of the receive buffer register at the time the reception of the next byte is complete, both the error interrupt request flag S0EIR and the overrun error status flag S0OE will be set, provided the overrun check has been enabled by bit S0OEN.

11.3 Hardware Error Detection Capabilities

To improve the safety of serial data exchange, the serial channel ASC0 provides an error interrupt request flag, which indicates the presence of an error, and three (selectable) error status flags in register S0CON, which indicate which error has been detected during reception. Upon completion of a reception, the error interrupt request flag S0EIR will be set simultaneously with the receive interrupt request flag S0RIR, if one or more of the following conditions are met:

•If the framing error detection enable bit S0FEN is set and any of the expected stop bits is not high, the framing error flag S0FE is set, indicating that the error interrupt request is due to a framing error (Asynchronous mode only).

•If the parity error detection enable bit S0PEN is set in the modes where a parity bit is received, and the parity check on the received data bits proves false, the parity error flag S0PE is set, indicating that the error interrupt request is due to a parity error (Asynchronous mode only).

•If the overrun error detection enable bit S0OEN is set and the last character received was not read out of the receive buffer by software or PEC transfer at the time the reception of a new frame is complete, the overrun error flag S0OE is set indicating that the error interrupt request is due to an overrun error (Asynchronous and synchronous mode).

11.4 ASC0 Baud Rate Generation

The serial channel ASC0 has its own dedicated 13-bit baud rate generator with 13-bit reload capability, allowing baud rate generation independent of the GPT timers.

The baud rate generator is clocked with the CPU clock divided by 2 ($f_{CPU}/2$). The timer is counting downwards and can be started or stopped through the Baud Rate Generator Run Bit S0R in register S0CON. Each underflow of the timer provides one clock pulse to the serial channel. The timer is reloaded with the value stored in its 13-bit reload register each time it underflows. The resulting clock is again divided according to the operating mode and controlled by the Baudrate Selection Bit S0BRS. If S0BRS='1', the clock signal is additionally divided to 2/3rd of its frequency (see formulas and table). So the baud rate of ASC0 is determined by the CPU clock, the reload value, the value of S0BRS and the operating mode (asynchronous or synchronous).

Register S0BG is the dual-function Baud Rate Generator/Reload register. Reading S0BG returns the content of the timer (bits 15...13 return zero), while writing to S0BG always updates the reload register (bits 15...13 are insiginificant).

An auto-reload of the timer with the content of the reload register is performed each time S0BG is written to. However, if S0R='0' at the time the write operation to S0BG is performed, the timer will not be reloaded until the first instruction cycle after S0R='1'.

Asynchronous Mode Baud Rates

For asynchronous operation, the baud rate generator provides a clock with 16 times the rate of the established baud rate. Every received bit is sampled at the 7th, 8th and 9th cycle of this clock. The baud rate for asynchronous operation of serial channel ASC0 and the required reload value for a given baudrate can be determined by the following formulas:

$$B_{Async} = \frac{f_{CPU}}{16 * (2 + \text{<S0BRS>}) * (\text{<S0BRL>} + 1)} \qquad \text{S0BRL} = \left(\frac{f_{CPU}}{16 * (2 + \text{<S0BRS>}) * B_{Async}}\right) - 1$$

<S0BRL> represents the content of the reload register, taken as unsigned 13-bit integer, <S0BRS> represents the value of bit S0BRS (i.e. ‘0’ or ‘1’), taken as integer.

The tables below list various commonly used baud rates together with the required reload values and the deviation errors compared to the intended baudrate for a number of CPU frequencies.

Note: The deviation errors given in the tables below are rounded. Using a baudrate crystal (e.g. 18.432 MHz) will provide correct baudrates without deviation errors.

Table 11-1 ASC0 Asynchronous Baudrate Generation for f_{CPU} = 25 MHz

Baud Rate	S0BRS = ‘0’		**S0BRS = ‘1’**	
	Deviation Error	**Reload Value**	**Deviation Error**	**Reload Value**
780 KBaud	+0.2 %	0000_H	---	---
19.2 KBaud	+1.7 % / -0.8 %	0027_H / 0028_H	+0.5 % / -3.1 %	$001A_H$ / $001B_H$
9600 Baud	+0.5 % / -0.8 %	0050_H / 0051_H	+0.5 % / -1.4 %	0035_H / 0036_H
4800 Baud	+0.5 % / -0.2 %	$00A1_H$ / $00A2_H$	+0.5 % / -0.5 %	$006B_H$ / $006C_H$
2400 Baud	+0.2 % / -0.2 %	0145_H / 0146_H	+0.0 % / -0.5 %	$00D8_H$ / $00D9_H$
1200 Baud	+0.0 % / -0.2 %	$028A_H$ / $028B_H$	+0.0 % / -0.2 %	$01B1_H$ / $01B2_H$
600 Baud	+0.0 % / -0.1 %	0515_H / 0516_H	+0.0 % / -0.1 %	0363_H / 0364_H
95 Baud	+0.4 %	$1FFF_H$	+0.0 % / -0.0 %	1569_H / $156A_H$
63 Baud	---	---	+1.0 %	$1FFF_H$

Table 11-2 ASC0 Asynchronous Baudrate Generation for f_{CPU} = 20 MHz

Baud Rate	**S0BRS = ‘0’**		**S0BRS = ‘1’**	
	Deviation Error	**Reload Value**	**Deviation Error**	**Reload Value**
625 KBaud	±0.0 %	0000_H	---	---
19.2 KBaud	+1.7 % / -1.4 %	$001F_H$ / 0020_H	+3.3 % / -1.4 %	0014_H / 0015_H
9600 Baud	+0.2 % / -1.4 %	0040_H / 0041_H	+1.0 % / -1.4 %	$002A_H$ / $002B_H$
4800 Baud	+0.2 % / -0.6 %	0081_H / 0082_H	+1.0 % / -0.2 %	0055_H / 0056_H
2400 Baud	+0.2 % / -0.2 %	0103_H / 0104_H	+0.4 % / -0.2 %	$00AC_H$ / $00AD_H$
1200 Baud	+0.2 % / -0.4 %	0207_H / 0208_H	+0.1 % / -0.2 %	$015A_H$ / $015B_H$
600 Baud	+0.1 % / -0.0 %	0410_H / 0411_H	+0.1 % / -0.1 %	$02B5_H$ / $02B6_H$
75 Baud	+1.7 %	$1FFF_H$	+0.0 % / -0.0 %	$15B2_H$ / $15B3_H$
50 Baud	---	---	+1.7 %	$1FFF_H$

Table 11-3 ASC0 Asynchronous Baudrate Generation for f_{CPU} = 16 MHz

Baud Rate	S0BRS = '0'		S0BRS = '1'	
	Deviation Error	**Reload Value**	**Deviation Error**	**Reload Value**
500 KBaud	±0.0 %	0000_H	---	---
19.2 KBaud	+0.2 % / -3.5 %	0019_H / $001A_H$	+2.1 % / -3.5 %	0010_H / 0011_H
9600 Baud	+0.2 % / -1.7 %	0033_H / 0034_H	+2.1 % / -0.8 %	0021_H / 0022_H
4800 Baud	+0.2 % / -0.8 %	0067_H / 0068_H	+0.6 % / -0.8 %	0044_H / 0045_H
2400 Baud	+0.2 % / -0.3 %	$00CF_H$ / $00D0_H$	+0.6 % / -0.1 %	0089_H / $008A_H$
1200 Baud	+0.4 % / -0.1 %	$019F_H$ / $01A0_H$	+0.3 % / -0.1 %	0114_H / 0115_H
600 Baud	+0.0 % / -0.1 %	0340_H / 0341_H	+0.1 % / -0.1 %	$022A_H$ / $022B_H$
61 Baud	+0.1 %	$1FFF_H$	+0.0 % / -0.0 %	$115B_H$ / $115C_H$
40 Baud	---	---	+1.7 %	$1FFF_H$

Synchronous Mode Baud Rates

For synchronous operation, the baud rate generator provides a clock with 4 times the rate of the established baud rate. The baud rate for synchronous operation of serial channel ASC0 can be determined by the following formula:

$$B_{Sync} = \frac{f_{CPU}}{4 * (2 + \text{<S0BRS>}) * (\text{<S0BRL>} + 1)} \qquad \text{S0BRL} = \left(\frac{f_{CPU}}{4 * (2 + \text{<S0BRS>}) * B_{Sync}}\right) - 1$$

<S0BRL> represents the content of the reload register, taken as unsigned 13-bit integers,
<S0BRS> represents the value of bit S0BRS (i.e. ‘0’ or ‘1’), taken as integer.

The table below gives the limit baudrates depending on the CPU clock frequency and bit S0BRS.

Table 11-4 ASC0 Synchronous Baudrate Generation

CPU clock f_{CPU}	S0BRS = ‘0’		S0BRS = ‘1’	
	Min. Baudrate	Max. Baudrate	Min. Baudrate	Max. Baudrate
16 MHz	244 Baud	2.000 MBaud	162 Baud	1.333 MBaud
20 MHz	305 Baud	2.500 MBaud	203 Baud	1.666 MBaud
25 MHz	381 Baud	3.125 MBaud	254 Baud	2.083 MBaud

11.5 ASC0 Interrupt Control

Four bit addressable interrupt control registers are provided for serial channel ASC0. Register S0TIC controls the transmit interrupt, S0TBIC controls the transmit buffer interrupt, S0RIC controls the receive interrupt and S0EIC controls the error interrupt of serial channel ASC0. Each interrupt source also has its own dedicated interrupt vector. S0TINT is the transmit interrupt vector, S0TBINT is the transmit buffer interrupt vector, S0RINT is the receive interrupt vector, and S0EINT is the error interrupt vector.

The cause of an error interrupt request (framing, parity, overrun error) can be identified by the error status flags in control register S0CON.

Note: In contrast to the error interrupt request flag S0EIR, the error status flags S0FE/S0PE/S0OE are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.

S0TIC
ASC0 Tx Intr. Ctrl. Reg. **SFR (FF6C$_H$/B6$_H$)** **Reset value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								S0 TIR	S0 TIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

S0TBIC
ASC0 Tx Buf. Intr. Ctrl. Reg. **SFR (FF9C$_H$/CE$_H$)** **Reset value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								S0 TBIR	S0 TBIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

S0RIC
ASC0 Rx Intr. Ctrl. Reg. **SFR (FF6E$_H$/B7$_H$)** **Reset value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								S0 RIR	S0 RIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

S0EIC

ASC0 Error Intr. Ctrl. Reg. **SFR (FF70$_H$/B8$_H$)** **Reset value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								S0 EIR	S0 EIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

Using the ASC0 Interrupts

For normal operation (i.e. besides the error interrupt) the ASC0 provides three interrupt requests to control data exchange via this serial channel:

- S0TBIR is activated when data is moved from S0TBUF to the transmit shift register.
- S0TIR is activated before the last bit of an asynchronous frame is transmitted, or after the last bit of a synchronous frame has been transmitted.
- S0RIR is activated when the received frame is moved to S0RBUF.

While the task of the receive interrupt handler is quite clear, the transmitter is serviced by two interrupt handlers. This provides advantages for the servicing software.

For single transfers is is sufficient to use the transmitter interrupt (S0TIR), which indicates that the previously loaded data has been transmitted, except for the last bit of an asynchronous frame.

For multiple back-to-back transfers it is necessary to load the following piece of data at last until the time the last bit of the previous frame has been transmitted. In asynchronous mode this leaves just one bit-time for the handler to respond to the transmitter interrupt request, in synchronous mode it is impossible at all.
Using the transmit buffer interrupt (S0TBIR) to reload transmit data gives the time to transmit a complete frame for the service routine, as S0TBUF may be reloaded while the previous data is still being transmitted.

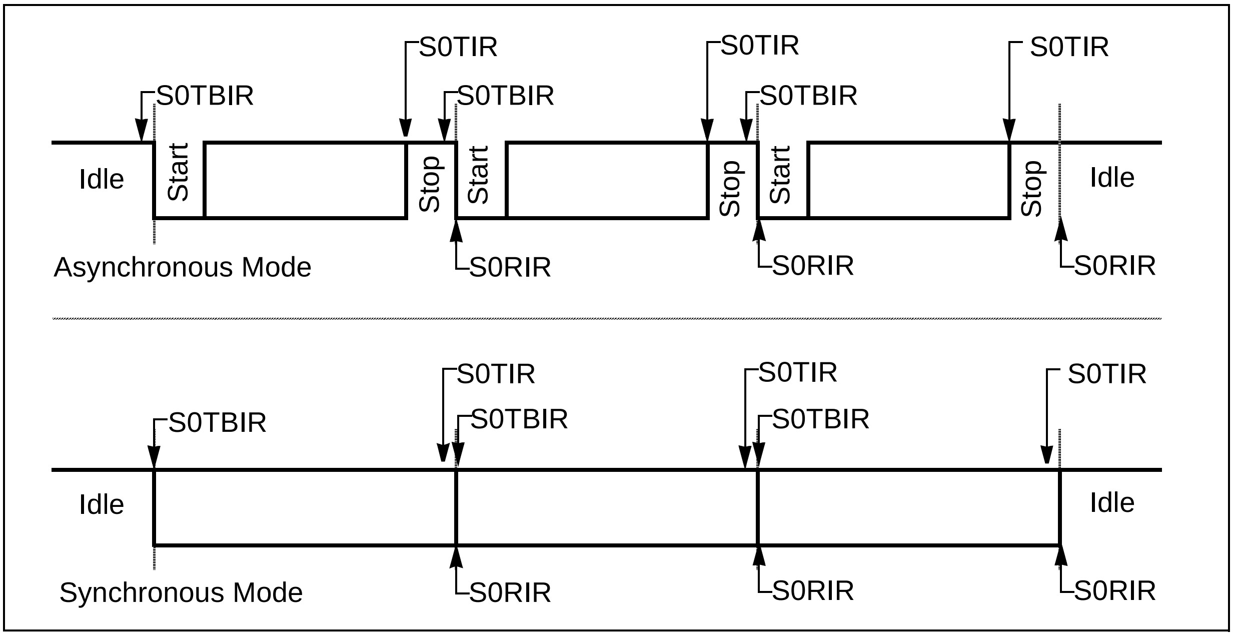


Figure 11-6 ASC0 Interrupt Generation

As shown in the figure above, S0TBIR is an early trigger for the reload routine, while S0TIR indicates the completed transmission. Software using handshake therefore should rely on S0TIR at the end of a data block to make sure that all data has really been transmitted.

12 The High-Speed Synchronous Serial Interface

The High-Speed Synchronous Serial Interface SSC provides flexible high-speed serial communication between the C164 and other microcontrollers, microprocessors or external peripherals.

The SSC supports full-duplex and half-duplex synchronous communication up to 6.25 MBaud (@ 25 MHz CPU clock). The serial clock signal can be generated by the SSC itself (master mode) or be received from an external master (slave mode). Data width, shift direction, clock polarity and phase are programmable. This allows communication with SPI-compatible devices. Transmission and reception of data is double-buffered. A 16-bit baud rate generator provides the SSC with a separate serial clock signal.

The high-speed synchronous serial interface can be configured in a very flexible way, so it can be used with other synchronous serial interfaces (e.g. the ASC0 in synchronous mode), serve for master/slave or multimaster interconnections or operate compatible with the popular SPI interface. So it can be used to communicate with shift registers (IO expansion), peripherals (e.g. EEPROMs etc.) or other controllers (networking). The SSC supports half-duplex and full-duplex communication. Data is transmitted or received on pins MTSR/P3.9 (Master Transmit / Slave Receive) and MRST/P3.8 (Master Receive / Slave Transmit). The clock signal is output or input on pin SCLK/P3.13. These pins are alternate functions of Port 3 pins.

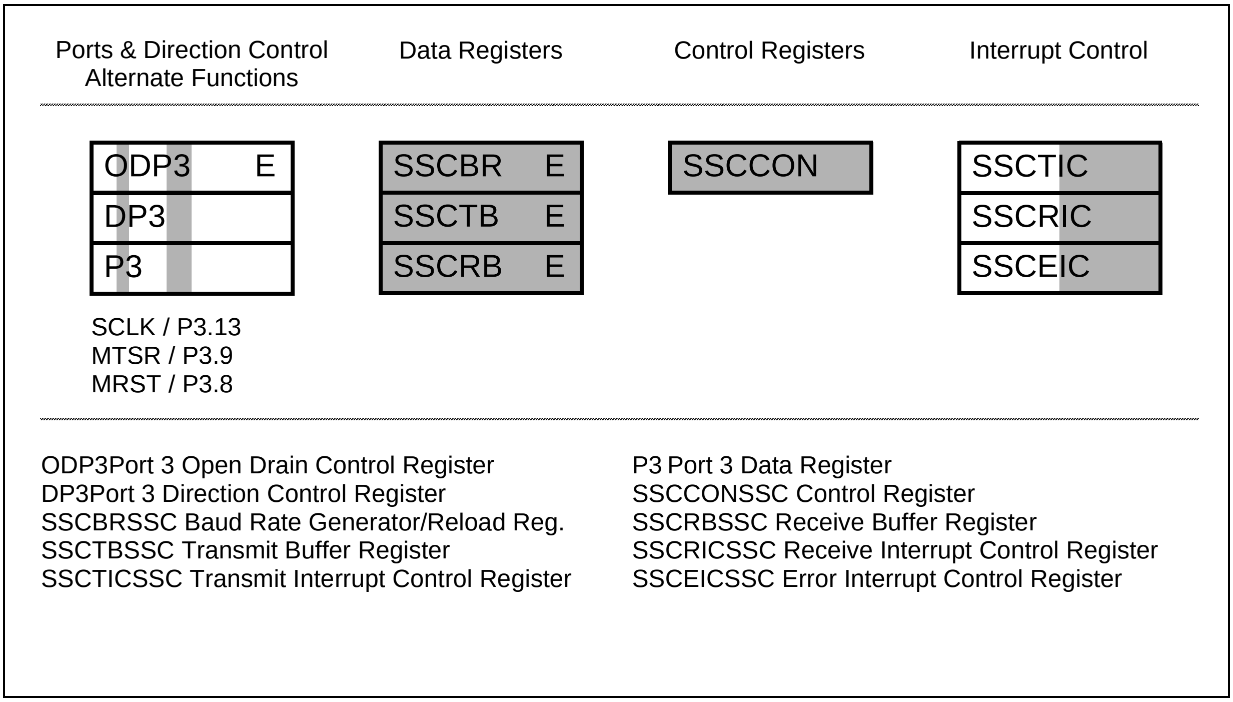

Ports & Direction Control
Alternate Functions
Data Registers
Control Registers
Interrupt Control
ODP3 E
DP3
P3
SSCBR E
SSCTB E
SSCRB E
SSCCON
SSCTIC
SSCRIC
SSCEIC
SCLK / P3.13
MTSR / P3.9
MRST / P3.8
ODP3Port 3 Open Drain Control Register
DP3Port 3 Direction Control Register
SSCBRSSC Baud Rate Generator/Reload Reg.
SSCTBSSC Transmit Buffer Register
SSCTICSSC Transmit Interrupt Control Register
P3 Port 3 Data Register
SSCCONSSC Control Register
SSCRBSSC Receive Buffer Register
SSCRICSSC Receive Interrupt Control Register
SSCEICSSC Error Interrupt Control Register

Figure 12-1 SFRs and Port Pins associated with the SSC

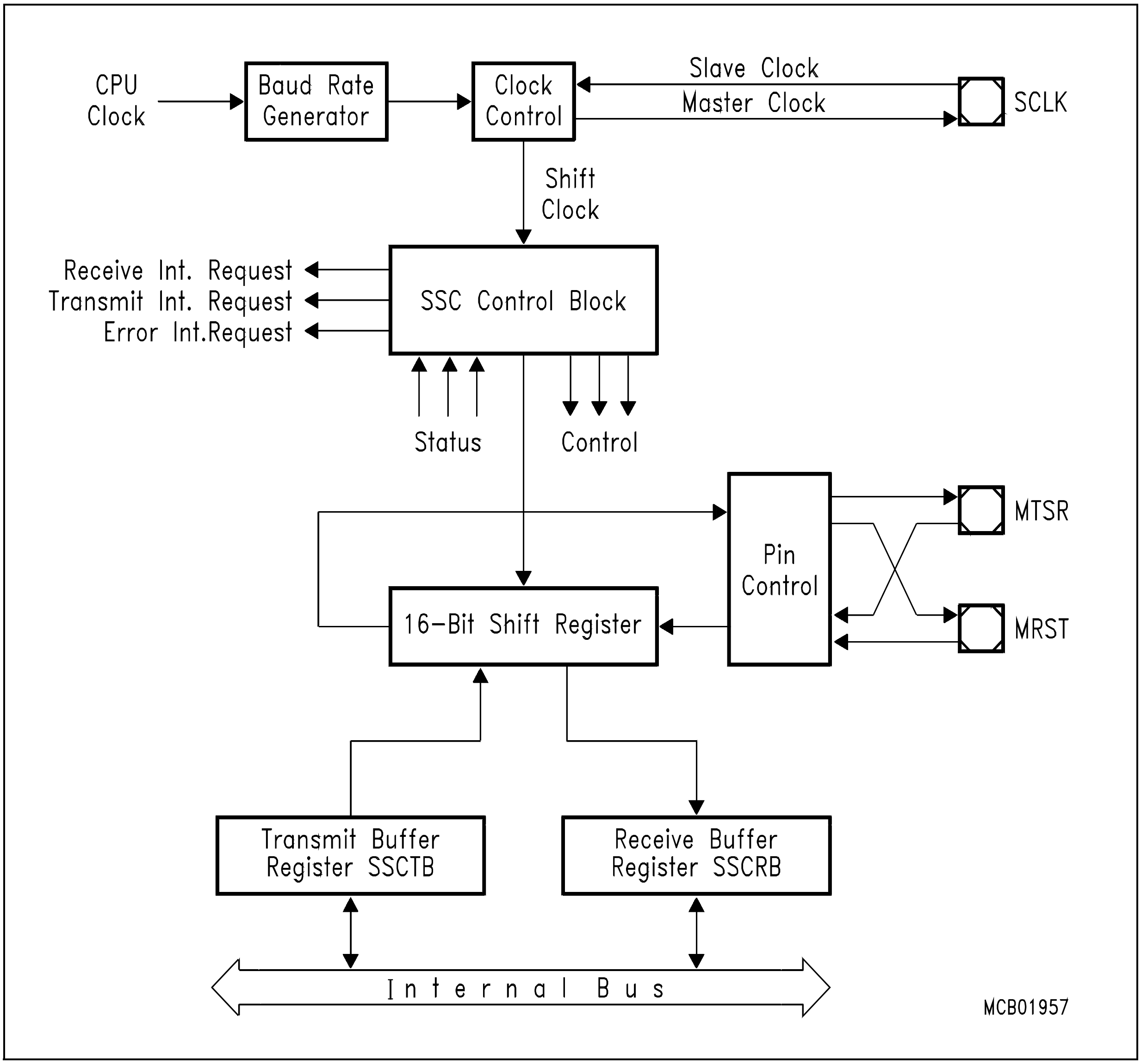


Figure 12-2 Synchronous Serial Channel SSC Block Diagram

The operating mode of the serial channel SSC is controlled by its bit-addressable control register SSCCON. This register serves for two purposes:

- during programming (SSC disabled by SSCEN='0') it provides access to a set of ctrl. bits,
- during operation (SSC enabled by SSCEN='1') it provides access to a set of status flags.

Register SSCCON is shown below in each of the two modes.

SSCCON

SSC Control Reg. (Pr.M.)   **SFR (FFB2_H/D9_H)**   **Reset value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC EN =0	**SSC MS**	**-**	**SSC AR EN**	**SSC BEN**	**SSC PEN**	**SSC REN**	**SSC TEN**	**-**	**SSC PO**	**SSC PH**	**SSC HB**	**SSCBM**			
rw	rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			

Bit	Function (Programming Mode, SSCEN = ‘0’)
SSCBM	**SSC Data Width Selection** 0: Reserved. Do not use this combination. 1..15: Transfer Data Width is 2...16 bit (<SSCBM>+1)
SSCHB	**SSC Heading Control Bit** 0: Transmit/Receive LSB First 1: Transmit/Receive MSB First
SSCPH	**SSC Clock Phase Control Bit** 0: Shift transmit data on the leading clock edge, latch on trailing edge 1: Latch receive data on leading clock edge, shift on trailing edge
SSCPO	**SSC Clock Polarity Control Bit** 0: Idle clock line is low, leading clock edge is low-to-high transition 1: Idle clock line is high, leading clock edge is high-to-low transition
SSCTEN	**SSC Transmit Error Enable Bit** 0: Ignore transmit errors 1: Check transmit errors
SSCREN	**SSC Receive Error Enable Bit** 0: Ignore receive errors 1: Check receive errors
SSCPEN	**SSC Phase Error Enable Bit** 0: Ignore phase errors 1: Check phase errors
SSCBEN	**SSC Baudrate Error Enable Bit** 0: Ignore baudrate errors 1: Check baudrate errors
SSCAREN	**SSC Automatic Reset Enable Bit** 0: No additional action upon a baudrate error 1: The SSC is automatically reset upon a baudrate error
SSCMS	**SSC Master Select Bit** 0: Slave Mode. Operate on shift clock received via SCLK. 1: Master Mode. Generate shift clock and output it via SCLK.
SSCEN	**SSC Enable Bit = ‘0’** Transmission and reception disabled. Access to control bits.

SSCCON

SSC Control Reg. (Op.M.) **SFR (FFB2$_H$/D9$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC EN =1	**SSC MS**	**-**	**SSC BSY**	**SSC BE**	**SSC PE**	**SSC RE**	**SSC TE**	**-**	**-**	**-**	**-**	**SSCBC**			
rw	rw	-	rwh	rwh	rwh	rwh	rwh	-	-	-	-	rh			

Bit	Function (Operating Mode, SSCEN = ‘1’)
SSCBC	**SSC Bit Count Field** Shift counter is updated with every shifted bit. **Do not write to!!!**
SSCTE	**SSC Transmit Error Flag** 1: Transfer starts with the slave’s transmit buffer not being updated
SSCRE	**SSC Receive Error Flag** 1: Reception completed before the receive buffer was read
SSCPE	**SSC Phase Error Flag** 1: Received data changes around sampling clock edge
SSCBE	**SSC Baudrate Error Flag** 1: More than factor 2 or less than factor 0.5 between Slave’s actual and expected baudrate
SSCBSY	**SSC Busy Flag** Set while a transfer is in progress. **Do not write to!!!**
SSCMS	**SSC Master Select Bit** 0: Slave Mode. Operate on shift clock received via SCLK. 1: Master Mode. Generate shift clock and output it via SCLK.
SSCEN	**SSC Enable Bit = ‘1’** Transmission and reception enabled. Access to status flags and M/S control.

Note: • *The target of an access to SSCCON (control bits or flags) is determined by the state of SSCEN prior to the access, i.e. writing C057$_H$ to SSCCON in programming mode (SSCEN=’0’) will initialize the SSC (SSCEN was ‘0’) and then turn it on (SSCEN=’1’).*

• *When writing to SSCCON, make sure that reserved locations receive zeros.*

The shift register of the SSC is connected to both the transmit pin and the receive pin via the pin control logic (see block diagram). Transmission and reception of serial data is synchronized and takes place at the same time, i.e. the number of transmitted bits is also received. Transmit data is written into the Transmit Buffer SSCTB. It is moved to the shift register as soon as this is empty. An SSC-master (SSCMS='1') immediately begins transmitting, while an SSC-slave (SSCMS='0') will wait for an active shift clock. When the transfer starts, the busy flag SSCBSY is set and a transmit interrupt request (SSCTIR) will be generated to indicate that SSCTB may be reloaded again. When the programmed number of bits (2...16) has been transferred, the contents of the shift register are moved to the Receive Buffer SSCRB and a receive interrupt request (SSCRIR) will be generated. If no further transfer is to take place (SSCTB is empty), SSCBSY will be cleared at the same time. Software should not modify SSCBSY, as this flag is hardware controlled.

Note: Only one SSC (etc.) can be master at a given time.

The transfer of serial data bits can be programmed in many respects:

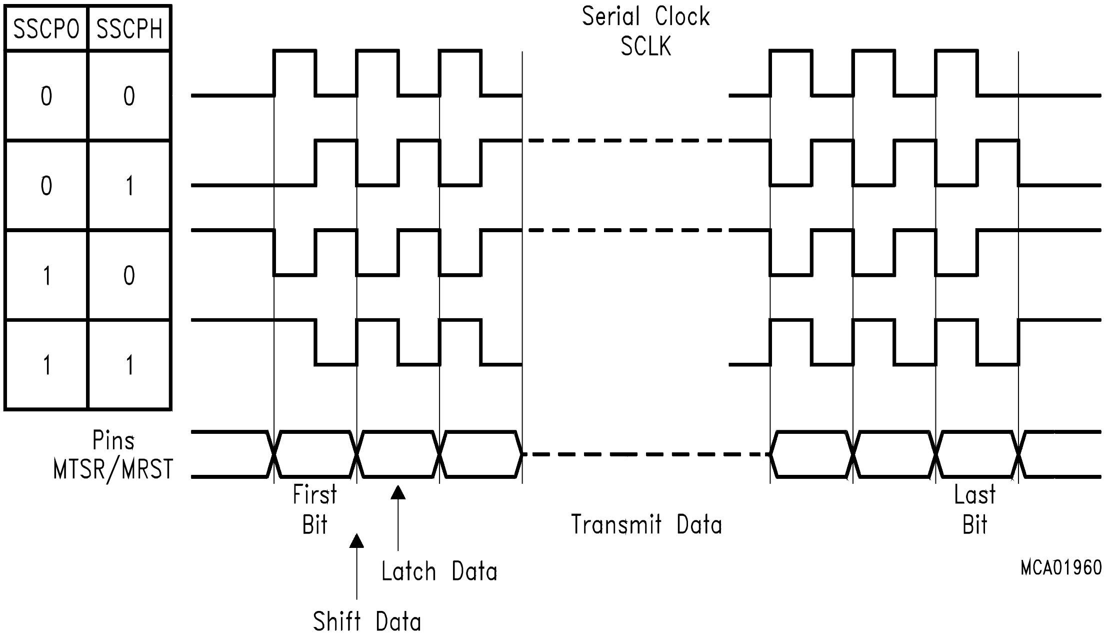
- the data width can be chosen from 2 bits to 16 bits
- transfer may start with the LSB or the MSB
- the shift clock may be idle low or idle high
- data bits may be shifted with the leading or trailing edge of the clock signal
- the baudrate may be set within a wide range (see baudrate generation)
- the shift clock can be generated (master) or received (slave)

This allows the adaptation of the SSC to a wide range of applications, where serial data transfer is required.

The Data Width Selection supports the transfer of frames of any length, from 2-bit "characters" up to 16-bit "characters". Starting with the LSB (SSCHB='0') allows communication e.g. with ASC0 devices in synchronous mode (C166 Family) or 8051 like serial interfaces. Starting with the MSB (SSCHB='1') allows operation compatible with the SPI interface.

Regardless which data width is selected and whether the MSB or the LSB is transmitted first, the transfer data is always right aligned in registers SSCTB and SSCRB, with the LSB of the transfer data in bit 0 of these registers. The data bits are rearranged for transfer by the internal shift register logic. The unselected bits of SSCTB are ignored, the unselected bits of SSCRB will be not valid and should be ignored by the receiver service routine.

The Clock Control allows the adaptation of transmit and receive behaviour of the SSC to a variety of serial interfaces. A specific clock edge (rising or falling) is used to shift out transmit data, while the other clock edge is used to latch in receive data. Bit SSCPH selects the leading edge or the trailing edge for each function. Bit SSCPO selects the level of the clock line in the idle state. So for an idle-high clock the leading edge is a falling one, a 1-to-0 transition. The figure below is a summary.



SSCPO	SSCPH
0	0
0	1
1	0
1	1

Figure 12-3 Serial Clock Phase and Polarity Options

12.1 Full-Duplex Operation

The different devices are connected through three lines. The definition of these lines is always determined by the master: The line connected to the master’s data output pin MTSR is the transmit line, the receive line is connected to its data input line MRST, and the clock line is connected to pin SCLK. Only the device selected for master operation generates and outputs the serial clock on pin SCLK. All slaves receive this clock, so their pin SCLK must be switched to input mode (DP3.13=’0’). The output of the master’s shift register is connected to the external transmit line, which in turn is connected to the slaves’ shift register input. The output of the slaves’ shift register is connected to the external receive line in order to enable the master to receive the data shifted out of the slave. The external connections are hard-wired, the function and direction of these pins is determined by the master or slave operation of the individual device.

Note: The shift direction shown in the figure applies for MSB-first operation as well as for LSB-first operation.

When initializing the devices in this configuration, select one device for master operation (SSCMS=’1’), all others must be programmed for slave operation (SSCMS=’0’). Initialization includes the operating mode of the device's SSC and also the function of the respective port lines (see “Port Control”).

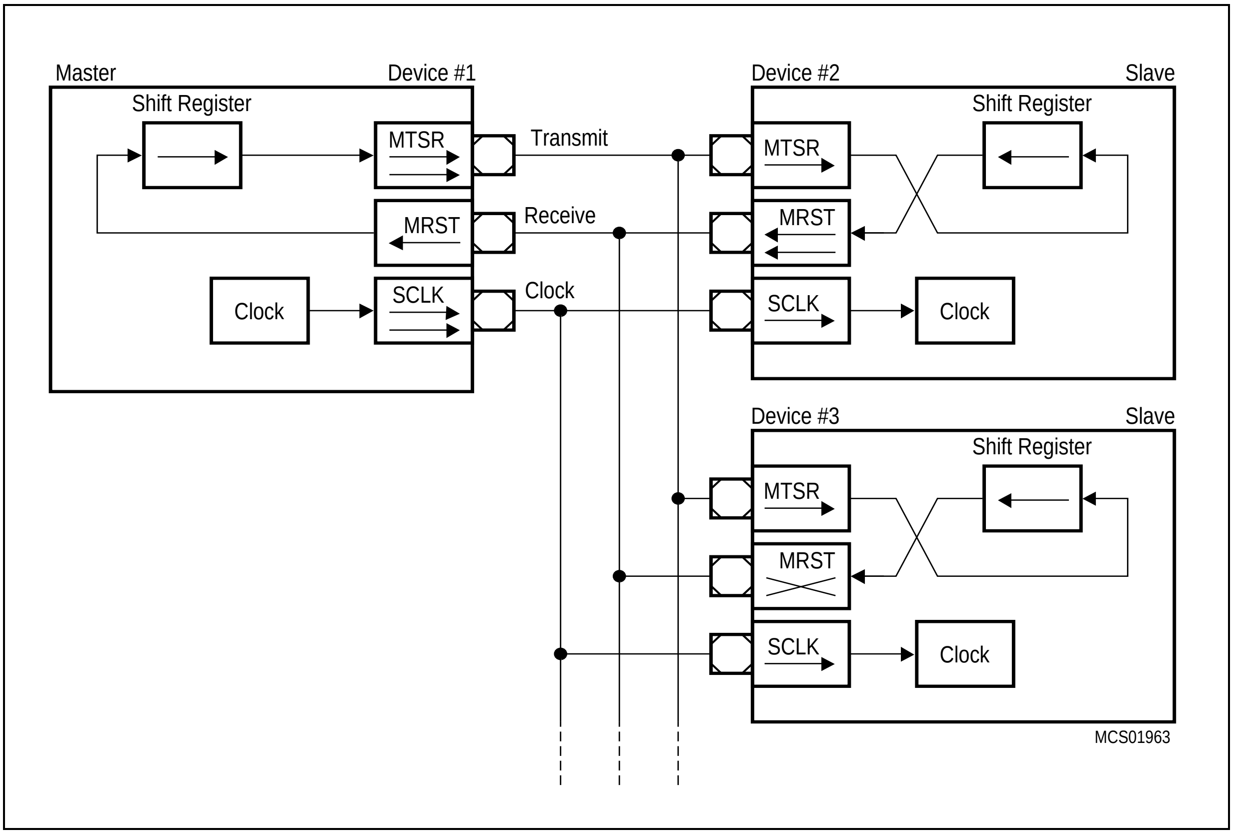


Figure 12-4 SSC Full Duplex Configuration

The data output pins MRST of all slave devices are connected together onto the one receive line in this configuration. During a transfer each slave shifts out data from its shift register. There are two ways to avoid collisions on the receive line due to different slave data:

Only one slave drives the line, i.e. enables the driver of its MRST pin. All the other slaves have to program their MRST pins to input. So only one slave can put its data onto the master's receive line. Only receiving of data from the master is possible. The master selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave. The selected slave then switches its MRST line to output, until it gets a deselection signal or command.

The slaves use open drain output on MRST. This forms a Wired-AND connection. The receive line needs an external pullup in this case. Corruption of the data on the receive line sent by the selected slave is avoided, when all slaves which are not selected for transmission to the master only send ones (‘1’). Since this high level is not actively driven onto the line, but only held through the pullup device, the selected slave can pull this line actively to a low level when transmitting a zero bit. The master selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave.

After performing all necessary initializations of the SSC, the serial interfaces can be enabled. For a master device, the alternate clock line will now go to its programmed polarity. The alternate data line will go to either '0' or '1', until the first transfer will start.

When the serial interfaces are enabled, the master device can initiate the first data transfer by writing the transmit data into register SSCTB. This value is copied into the shift register (which is assumed to be empty at this time), and the selected first bit of the transmit data will be placed onto the MTSR line on the next clock from the baudrate generator (transmission only starts, if SSCEN=’1’). Depending on the selected clock phase, also a clock pulse will be generated on the SCLK line. With the opposite clock edge the master at the same time latches and shifts in the data detected at its input line MRST. This “exchanges” the transmit data with the receive data. Since the clock line is connected to all slaves, their shift registers will be shifted synchronously with the master's shift register, shifting out the data contained in the registers, and shifting in the data detected at the input line. After the preprogrammed number of clock pulses (via the data width selection) the data transmitted by the master is contained in all slaves’ shift registers, while the master's shift register holds the data of the selected slave. In the master and all slaves the content of the shift register is copied into the receive buffer SSCRB and the receive interrupt flag SSCRIR is set.

A slave device will immediately output the selected first bit (MSB or LSB of the transfer data) at pin MRST, when the content of the transmit buffer is copied into the slave's shift register. It will not wait for the next clock from the baudrate generator, as the master does. The reason for this is that, depending on the selected clock phase, the first clock edge generated by the master may be already used to clock in the first data bit. So the slave's first data bit must already be valid at this time.

Note: On the SSC always a transmission ***and*** *a reception takes place at the same time, regardless whether valid data has been transmitted or received. This is different e.g. from asynchronous reception on ASC0.*

The initialization of the SCLK pin on the master requires some attention in order to avoid undesired clock transitions, which may disturb the other receivers. The state of the internal alternate output lines is '1' as long as the SSC is disabled. This alternate output signal is ANDed with the respective port line output latch. Enabling the SSC with an idle-low clock (SSCPO='0') will drive the alternate data output and (via the AND) the port pin SCLK immediately low. To avoid this, use the following sequence:

- select the clock idle level (SSCPO='x')
- load the port output latch with the desired clock idle level (P3.13='x')
- switch the pin to output (DP3.13='1')
- enable the SSC (SSCEN='1')
- if SSCPO='0': enable alternate data output (P3.13='1')

The same mechanism as for selecting a slave for transmission (separate select lines or special commands) may also be used to move the role of the master to another device in the network. In this case the previous master and the future master (previous slave) will have to toggle their operating mode (SSCMS) and the direction of their port pins (see description above).

12.2 Half Duplex Operation

In a half duplex configuration only one data line is necessary for both receiving **and** transmitting of data. The data exchange line is connected to both pins MTSR and MRST of each device, the clock line is connected to the SCLK pin.

The master device controls the data transfer by generating the shift clock, while the slave devices receive it. Due to the fact that all transmit and receive pins are connected to the one data exchange line, serial data may be moved between arbitrary stations.

Similar to full duplex mode there are **two ways to avoid collisions** on the data exchange line:

- only the transmitting device may enable its transmit pin driver
- the non-transmitting devices use open drain output and only send ones.

Since the data inputs and outputs are connected together, a transmitting device will clock in its own data at the input pin (MRST for a master device, MTSR for a slave). By these means any corruptions on the common data exchange line are detected, where the received data is not equal to the transmitted data.

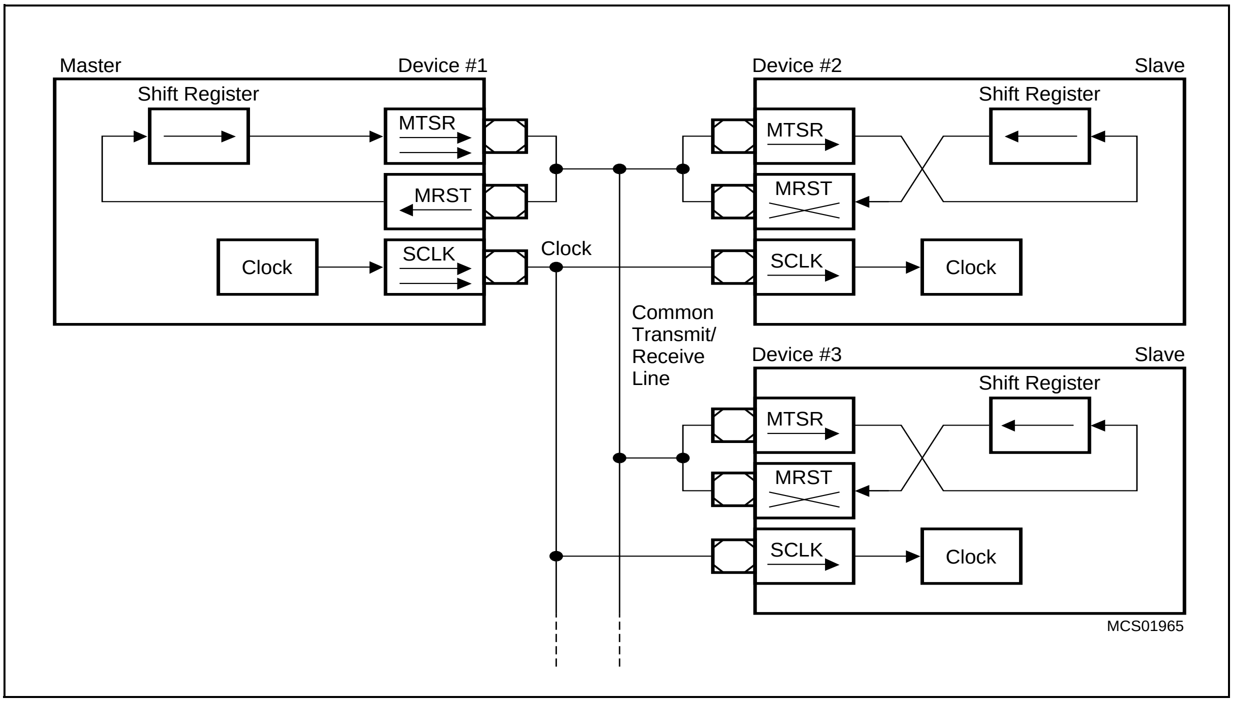


Figure 12-5 SSC Half Duplex Configuration

12.3 Continuous Transfers

When the transmit interrupt request flag is set, it indicates that the transmit buffer SSCTB is empty and ready to be loaded with the next transmit data. If SSCTB has been reloaded by the time the current transmission is finished, the data is immediately transferred to the shift register and the next transmission will start without any additional delay. On the data line there is no gap between the two successive frames. E.g. two byte transfers would look the same as one word transfer. This feature can be used to interface with devices which can operate with or require more than 16 data bits per transfer. It is just a matter of software, how long a total data frame length can be. This option can also be used e.g. to interface to byte-wide and word-wide devices on the same serial bus.

Note: Of course, this can only happen in multiples of the selected basic data width, since it would require disabling/enabling of the SSC to reprogram the basic data width on-the-fly.

12.4 Port Control

The SSC uses three pins of Port 3 to communicate with the external world. Pin P3.13/ SCLK serves as the clock line, while pins P3.8/MRST (Master Receive / Slave Transmit) and P3.9/MTSR (Master Transmit / Slave Receive) serve as the serial data input/output lines.The operation of these pins depends on the selected operating mode (master or slave). In order to enable the alternate output functions of these pins instead of the general purpose IO operation, the respective port latches have to be set to ’1’, since the port latch outputs and the alternate output lines are ANDed. When an alternate data output line is not used (function disabled), it is held at a high level, allowing IO operations via the port latch. The direction of the port lines depends on the operating mode. The SSC will automatically use the correct alternate input or output line of the ports when switching modes. The direction of the pins, however, must be programmed by the user, as shown in the tables. Using the open drain output feature helps to avoid bus contention problems and reduces the need for hardwired hand-shaking or slave select lines. In this case it is not always necessary to switch the direction of a port pin. The table below summarizes the required values for the different modes and pins.

SSC Port Control

Pin	Master Mode			Slave Mode		
	Function	**Port Latch**	**Direction**	**Function**	**Port Latch**	**Direction**
SCLK	Serial Clock Output	P3.13 = ’1’	DP3.13=’ 1’	Serial Clock Input	P3.13 = ’x’	DP3.13=’ 0’
MTSR	Serial Data Output	P3.9 = ’1’	DP3.9 = ’1’	Serial Data Input	P3.9 = ’x’	DP3.9 = ’0’
MRST	Serial Data Input	P3.8 = ’x’	DP3.8 = ’0’	Serial Data Output	P3.8 = ’1’	DP3.8 = ’1’

Note: In the table above, an ’x’ means that the actual value is irrelevant in the respective mode, however, it is recommended to set these bits to ’1’, so they are already in the correct state when switching between master and slave mode.

12.5 Baud Rate Generation

The serial channel SSC has its own dedicated 16-bit baud rate generator with 16-bit reload capability, permitting baud rate generation independent from the timers.

The baud rate generator is clocked with the CPU clock divided by 2 ($f_{CPU}/2$). The timer is counting downwards and can be started or stopped through the global enable bit SSCEN in register SSCCON. Register SSCBR is the dual-function Baud Rate Generator/Reload register. Reading SSCBR, while the SSC is enabled, returns the content of the timer. Reading SSCBR, while the SSC is disabled, returns the programmed reload value. In this mode the desired reload value can be written to SSCBR.

Note: Never write to SSCBR, while the SSC is enabled.

The formulas below calculate either the resulting baud rate for a given reload value, or the required reload value for a given baudrate:

$$B_{SSC} = \frac{f_{CPU}}{2 * (<SSCBR> + 1)} \qquad SSCBR = \left(\frac{f_{CPU}}{2 * \text{Baudrate}_{SSC}} \right) - 1$$

<SSCBR> represents the content of the reload register, taken as an unsigned 16-bit integer.

The table below lists some possible baud rates together with the required reload values and the resulting bit times, for different CPU clock frequencies.

Table 12-1 SSC Baudrate Calculations

Baud Rate for f_{CPU} = ...			Bit Time for f_{CPU} = ...			Reload Value (SSCBR)
16 MHz	20 MHz	25 MHz	16 MHz	20 MHz	25 MHz	
Reserved. SSCBR must be > 0.			Reserved. SSCBR must be > 0.			0000_H
4.00 MBaud	5.00 MBaud	6.25 MBaud	250 ns	200 ns	160 ns	0001_H
2.67 MBaud	3.33 MBaud	4.17 MBaud	375 ns	300 ns	240 ns	0002_H
2.00 MBaud	2.50 MBaud	3.13 MBaud	500 ns	400 ns	320 ns	0003_H
1.60 MBaud	2.00 MBaud	2.50 MBaud	625 ns	500 ns	400 ns	0004_H
1.00 MBaud	1.25 MBaud	1.56 MBaud	1.00 µs	800 ns	640 ns	0007_H
800 KBaud	1.0 MBaud	1.25 MBaud	1.25 µs	1 µs	800 ns	0009_H
100 KBaud	125 KBaud	156 KBaud	10 µs	8 µs	6.4 µs	$004F_H$
80 KBaud	100 KBaud	125 KBaud	12.5 µs	10 µs	8 µs	0063_H
64 KBaud	80 KBaud	100 KBaud	15.6 µs	12.5 µs	10 µs	$007C_H$
1.0 KBaud	1.25 KBaud	1.56 KBaud	1 ms	800 µs	640 µs	$1F3F_H$

Table 12-1 SSC Baudrate Calculations (cont'd)

Baud Rate for f_{CPU} = ...			Bit Time for f_{CPU} = ...			Reload Value (SSCBR)
16 MHz	**20 MHz**	**25 MHz**	**16 MHz**	**20 MHz**	**25 MHz**	
800 Baud	1.0 KBaud	1.25 KBaud	1.25 ms	1 ms	800 µs	$270F_H$
640 Baud	800 Baud	1.0 KBaud	1.56 ms	1.25 ms	1 ms	$30D3_H$
122.1 Baud	152.6 Baud	190.7 Baud	8.2 ms	6.6 ms	5.2 ms	$FFFF_H$

12.6 Error Detection Mechanisms

The SSC is able to detect four different error conditions. Receive Error and Phase Error are detected in all modes, while Transmit Error and Baudrate Error only apply to slave mode. When an error is detected, the respective error flag is set. When the corresponding Error Enable Bit is set, also an error interrupt request will be generated by setting SSCEIR (see figure below). The error interrupt handler may then check the error flags to determine the cause of the error interrupt. The error flags are not reset automatically (like SSCEIR), but rather must be cleared by software after servicing. This allows servicing of some error conditions via interrupt, while the others may be polled by software.

Note: The error interrupt handler must clear the associated (enabled) errorflag(s) to prevent repeated interrupt requests.

A **Receive Error** (Master or Slave mode) is detected, when a new data frame is completely received, but the previous data was not read out of the receive buffer register SSCRB. This condition sets the error flag SSCRE and, when enabled via SSCREN, the error interrupt request flag SSCEIR. The old data in the receive buffer SSCRB will be overwritten with the new value and is unretrievably lost.

A **Phase Error** (Master or Slave mode) is detected, when the incoming data at pin MRST (master mode) or MTSR (slave mode), sampled with the same frequency as the CPU clock, changes between one sample before and two samples after the latching edge of the clock signal (see "Clock Control"). This condition sets the error flag SSCPE and, when enabled via SSCPEN, the error interrupt request flag SSCEIR.

A **Baud Rate Error** (Slave mode) is detected, when the incoming clock signal deviates from the programmed baud rate by more than 100%, i.e. it either is more than double or less than half the expected baud rate. This condition sets the error flag SSCBE and, when enabled via SSCBEN, the error interrupt request flag SSCEIR. Using this error detection capability requires that the slave's baud rate generator is programmed to the same baud rate as the master device. This feature detects false additional, or missing pulses on the clock line (within a certain frame).

Note: If this error condition occurs and bit SSCAREN='1', an automatic reset of the SSC will be performed in case of this error. This is done to reinitialize the SSC, if too few or too many clock pulses have been detected.

A **Transmit Error** (Slave mode) is detected, when a transfer was initiated by the master (shift clock gets active), but the transmit buffer SSCTB of the slave was not updated since the last transfer. This condition sets the error flag SSCTE and, when enabled via SSCTEN, the error interrupt request flag SSCEIR. If a transfer starts while the transmit buffer is not updated, the slave will shift out the 'old' contents of the shift register, which normally is the data received during the last transfer.
This may lead to the corruption of the data on the transmit/receive line in half-duplex mode (open drain configuration), if this slave is not selected for transmission. This mode requires that slaves not selected for transmission only shift out ones, i.e. their transmit buffers must be loaded with 'FFFF$_H$' prior to any transfer.

Note: A slave with push/pull output drivers, which is not selected for transmission, will normally have its output drivers switched. However, in order to avoid possible conflicts or misinterpretations, it is recommended to always load the slave's transmit buffer prior to any transfer.

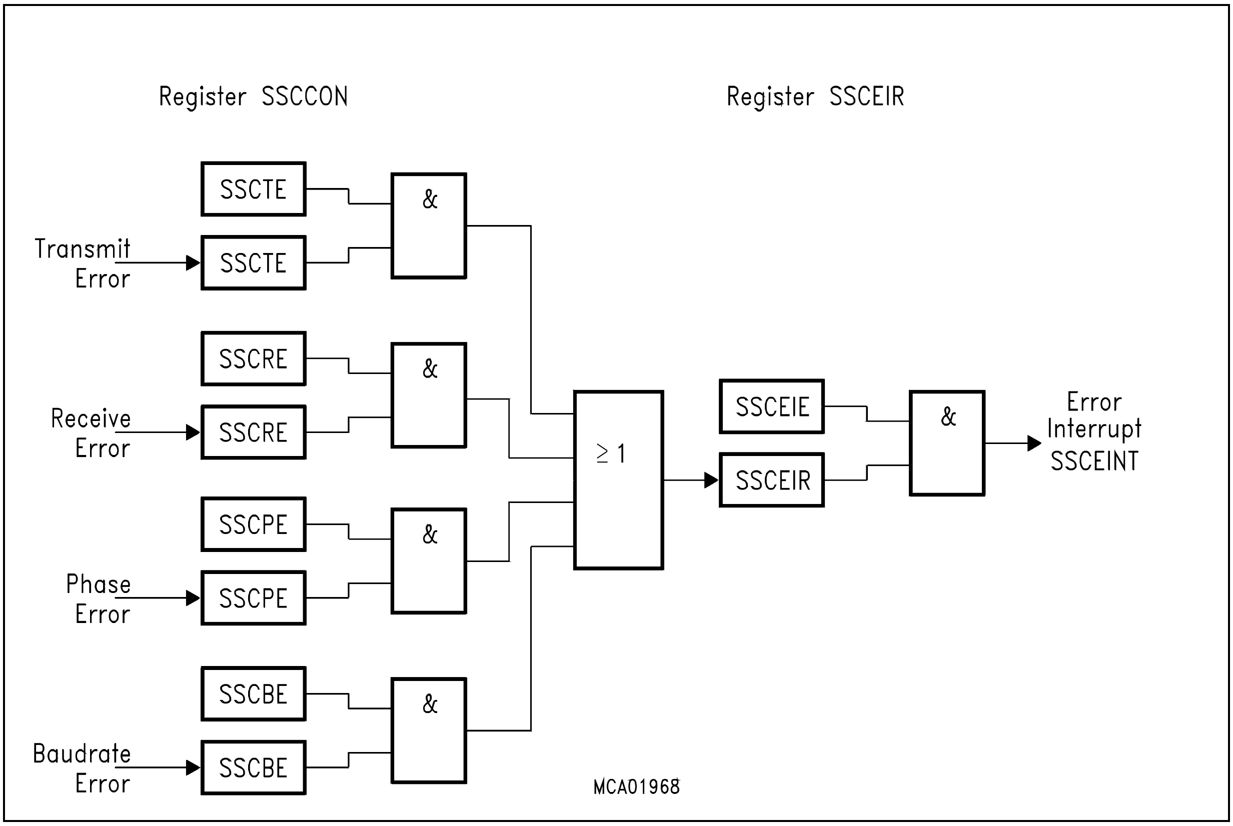


Figure 12-6 SSC Error Interrupt Control

12.7 SSC Interrupt Control

Three bit addressable interrupt control registers are provided for serial channel SSC. Register SSCTIC controls the transmit interrupt, SSCRIC controls the receive interrupt and SSCEIC controls the error interrupt of serial channel SSC. Each interrupt source also has its own dedicated interrupt vector. SCTINT is the transmit interrupt vector, SCRINT is the receive interrupt vector, and SCEINT is the error interrupt vector.

The cause of an error interrupt request (receive, phase, baudrate,transmit error) can be identified by the error status flags in control register SSCCON.

Note: In contrary to the error interrupt request flag SSCEIR, the error status flags SSCxE are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.

SSCTIC
SSC Transmit Intr. Ctrl. Reg. **SFR (FF72$_H$/B9$_H$)** **Reset value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								SSC TIR	SSC TIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

SSCRIC
SSC Receive Intr. Ctrl. Reg. **SFR (FF74$_H$/BA$_H$)** **Reset value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								SSC RIR	SSC RIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

SSCEIC
SSC Error Intr. Ctrl. Reg. **SFR (FF76$_H$/BB$_H$)** **Reset value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								SSC EIR	SSC EIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

13 The Watchdog Timer (WDT)

To allow recovery from software or hardware failure, the C164 provides a Watchdog Timer. If the software fails to service this timer before an overflow occurs, an internal reset sequence will be initiated. This internal reset will also pull the $\overline{\text{RSTOUT}}$ pin low, which also resets the peripheral hardware which might be the cause for the malfunction. When the watchdog timer is enabled and the software has been designed to service it regularly before it overflows, the watchdog timer will supervise the program execution as it only will overflow if the program does not progress properly. The watchdog timer will also time out if a software error was due to hardware related failures. This prevents the controller from malfunctioning for longer than a user-specified time.

Note: When the bidirectional reset is enabled also pin $\overline{\text{RSTIN}}$ will be pulled low for the duration of the internal reset sequence upon a software reset or a watchdog timer reset.

The watchdog timer provides two registers:

- a read-only timer register that contains the current count, and
- a control register for initialization and reset source detection.

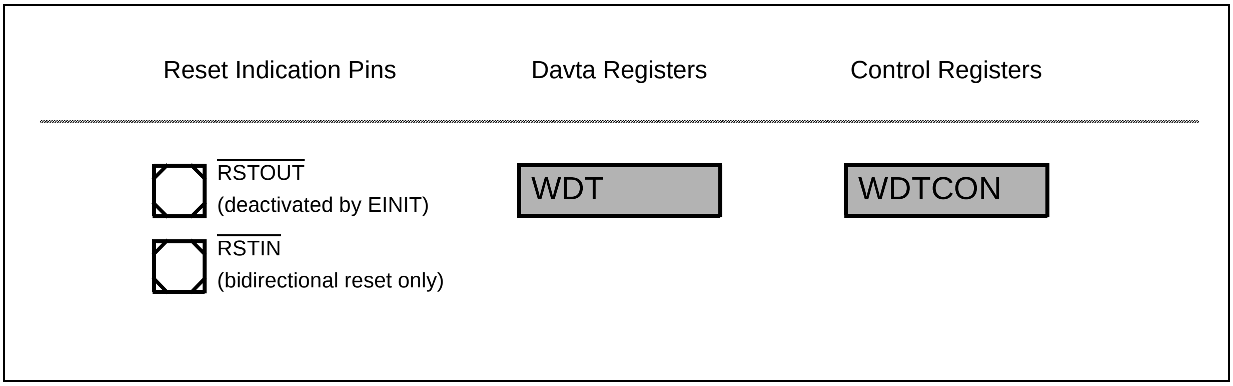


Figure 13-1 SFRs and Port Pins associated with the Watchdog Timer

The watchdog timer is a 16-bit up counter which is clocked with the prescaled CPU clock (f_{CPU}). The prescaler divides the CPU clock...

- by 2 (WDTIN = '0', WDTPRE = '0'), or
- by 4 (WDTIN = '0', WDTPRE = '1'), or
- by 128 (WDTIN = '1', WDTPRE = '0'), or
- by 256 (WDTIN = '1', WDTPRE = '1').

The 16-bit watchdog timer is realized as two concatenated 8-bit timers (see figure below). The upper 8 bits of the watchdog timer can be preset to a user-programmable value via a watchdog service access in order to vary the watchdog expire time. The lower 8 bits are reset upon each service access.

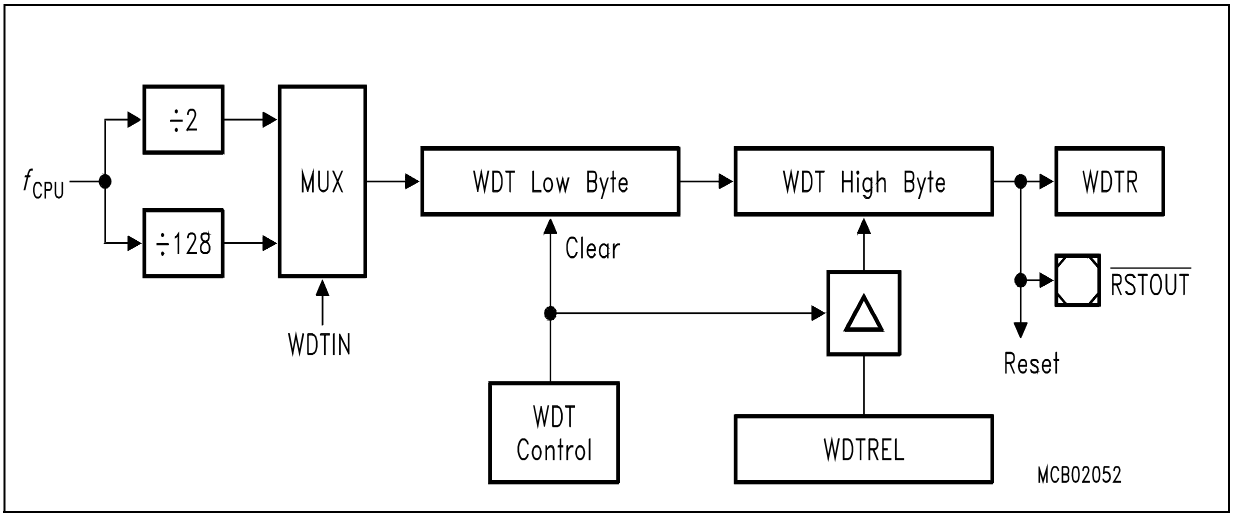


Figure 13-2 Watchdog Timer Block Diagram

13.1 Operation of the Watchdog Timer

The current count value of the Watchdog Timer is contained in the Watchdog Timer Register WDT which is a non-bitaddressable read-only register. The operation of the Watchdog Timer is controlled by its bitaddressable Watchdog Timer Control Register WDTCON. This register specifies the reload value for the high byte of the timer, selects the input clock prescaling factor and also provides flags that indicate the source of a reset.

After any reset (except see note) the watchdog timer is enabled and starts counting up from 0000_H with the default frequency $f_{WDT} = f_{CPU}/2$. The default input frequency may be changed to another frequency ($f_{WDT} = f_{CPU}/128$) by programming the prescaler (bit WDTIN).

The watchdog timer can be disabled by executing the instruction DISWDT (Disable Watchdog Timer). Instruction DISWDT is a protected 32-bit instruction which will ONLY be executed during the time between a reset and execution of either the EINIT (End of Initialization) or the SRVWDT (Service Watchdog Timer) instruction. Either one of these instructions disables the execution of DISWDT.

Note: After a hardware reset that activates the Bootstrap Loader the watchdog timer will be disabled. The software reset that terminates the BSL mode will then enable the WDT.

When the watchdog timer is not disabled via instruction DISWDT it will continue counting up, even during Idle Mode. If it is not serviced via the instruction SRVWDT by the time the count reaches $FFFF_H$ the watchdog timer will overflow and cause an internal reset. This reset will pull the external reset indication pin $\overline{\text{RSTOUT}}$ low. The Watchdog Timer Reset Indication Flag (WDTR) in register WDTCON will be set in this case.
In bidirectional reset mode also pin $\overline{\text{RSTIN}}$ will be pulled low for the duration of the internal reset sequence and a long hardware reset will be indicated instead.

A watchdog reset will also complete a running external bus cycle before starting the internal reset sequence if this bus cycle does not use $\overline{\text{READY}}$ or samples $\overline{\text{READY}}$ active (low) after the programmed waitstates. Otherwise the external bus cycle will be aborted.

To prevent the watchdog timer from overflowing it must be serviced periodically by the user software. The watchdog timer is serviced with the instruction SRVWDT which is a protected 32-bit instruction. Servicing the watchdog timer clears the low byte and reloads the high byte of the watchdog timer register WDT with the preset value from bitfield WDTREL which is the high byte of register WDTCON. Servicing the watchdog timer will also reset bit WDTR. After being serviced the watchdog timer continues counting up from the value (<WDTREL> * 2^8).

Instruction SRVWDT has been encoded in such a way that the chance of unintentionally servicing the watchdog timer (e.g. by fetching and executing a bit pattern from a wrong location) is minimized. When instruction SRVWDT does not match the format for protected instructions the Protection Fault Trap will be entered, rather than the instruction be executed.

WDTCON
WDT Control Register **SFR (FFAE_H/D7_H)** **Reset value: 00XX_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDTREL								**WDTPRE**	**-**	**-**	**LHWR**	**SHWR**	**SWR**	**WDTR**	**WDTIN**
-								rw	-	-	rh	rh	rh	rh	rw

Bit	Function
WDTIN	**Watchdog Timer Input Frequency Select** (combined with WDTPRE) Controls the input clock prescaler. See table below.
WDTR	**Watchdog Timer Reset Indication Flag** Cleared by a hardware reset or by the SRVWDT instruction.
SWR	**Software Reset Indication Flag**
SHWR	**Short Hardware Reset Indication Flag**
LHWR	**Long Hardware Reset Indication Flag**
WDTPRE	**Watchdog Timer Input Prescaler Control (combined with WDTIN)** Controls the input clock prescaler. See table below.
WDTREL	**Watchdog Timer Reload Value** (for the high byte of WDT)

Note: The reset value depends on the reset source (see description below).
The execution of EINIT clears the reset indication flags.

The time period for an overflow of the watchdog timer is programmable in two ways:

- **the input frequency** to the watchdog timer can be selected via a prescaler controlled by bits WDTPRE and WDTIN in register WDTCON to be $f_{\text{CPU}}/2$, $f_{\text{CPU}}/4$, $f_{\text{CPU}}/128$, or $f_{\text{CPU}}/256$.
- **the reload value** WDTREL for the high byte of WDT can be programmed in register WDTCON.

The period P_{WDT} between servicing the watchdog timer and the next overflow can therefore be determined by the following formula:

$$P_{WDT} = \frac{2^{(1\ +\ <WDTPRE>\ +\ <WDTIN>*6)} * (2^{16} - <WDTREL>*2^{8})}{f_{CPU}}$$

The table below marks the possible ranges (depending on the prescaler bits WDTIN and WDTPRE) for the watchdog time which can be achieved using a certain CPU clock.

Table 13-1 Watchdog Time Ranges

CPU clock f_{CPU}	Prescaler			Reload value in WDTREL		
	WDT IN	**WDT PRE**	f_{WDT}	**FF$_H$**	**7F$_H$**	**00$_H$**
12 MHz	0	0	f_{CPU} / 2	42.67 µs	5.50 ms	10.92 ms
	0	1	f_{CPU} / 4	85.33 µs	11.01 ms	21.85 ms
	1	0	f_{CPU} / 128	2.73 ms	352.3 ms	699.1 ms
	1	1	f_{CPU} / 256	5.46 ms	704.5 ms	1398 ms
16 MHz	0	0	f_{CPU} / 2	32.00 µs	4.13 ms	8.19 ms
	0	1	f_{CPU} / 4	64.00 µs	8.26 ms	16.38 ms
	1	0	f_{CPU} / 128	2.05 ms	264.2 ms	524.3 ms
	1	1	f_{CPU} / 256	4.10 ms	528.4 ms	1049 ms
20 MHz	0	0	f_{CPU} / 2	25.60 µs	3.30 ms	6.55 ms
	0	1	f_{CPU} / 4	51.20 µs	6.61 ms	13.11 ms
	1	0	f_{CPU} / 128	1.64 ms	211.4 ms	419.4 ms
	1	1	f_{CPU} / 256	3.28 ms	422.7 ms	838.9 ms
25 MHz	0	0	f_{CPU} / 2	20.48 µs	2.64 ms	5.24 ms
	0	1	f_{CPU} / 4	40.96 µs	5.28 ms	10.49 ms
	1	0	f_{CPU} / 128	1.31 ms	169.1 ms	335.5 ms
	1	1	f_{CPU} / 256	2.62 ms	338.2 ms	671.1 ms

Note: For safety reasons, the user is advised to rewrite WDTCON each time before the watchdog timer is serviced.

13.2 Reset Source Indication

The reset indication flags in register WDTCON provide information on the source for the last reset. As the C164 starts executing from location $00'0000_H$ after any possible reset event the initialization software may check these flags in order to determine if the recent reset event was triggered by an external hardware signal (via $\overline{\text{RSTIN}}$), by software itself or by an overflow of the watchdog timer. The initialization (and also the further operation) of the microcontroller system can thus be adapted to the respective circumstances, e.g. a special routine may verify the software integrity after a watchdog timer reset.

The reset indication flags are not mutually exclusive, i.e. more than one flag may be set after reset depending on its source. The table below summarizes the possible combinations:

Table 13-2 Reset Indication Flag Combinations

	Reset Indication Flags[1]			
Event	**LHWR**	**SHWR**	**SWR**	**WDTR**
Long Hardware Reset	1	1	1	0
Short Hardware Reset	*	1	1	0
Software Reset	*	*	1	-
Watchdog Timer Reset	*	*	1	1
EINIT instruction	0	0	0	-
SRVWDT instruction	-	-	-	0

[1] Description of table entries:
’1’ = flag is set, ’0’ = flag is cleared, ’-’ = flag is not affected,
’*’ = flag is set in bidirectional reset mode, not affected otherwise.

Long Hardware Reset is indicated when the $\overline{\text{RSTIN}}$ input is still sampled low (active) at the end of a hardware triggered internal reset sequence.

Short Hardware Reset is indicated when the $\overline{\text{RSTIN}}$ input is sampled high (inactive) at the end of a hardware triggered internal reset sequence.

Software Reset is indicated after a reset triggered by the excution of instruction SRST.

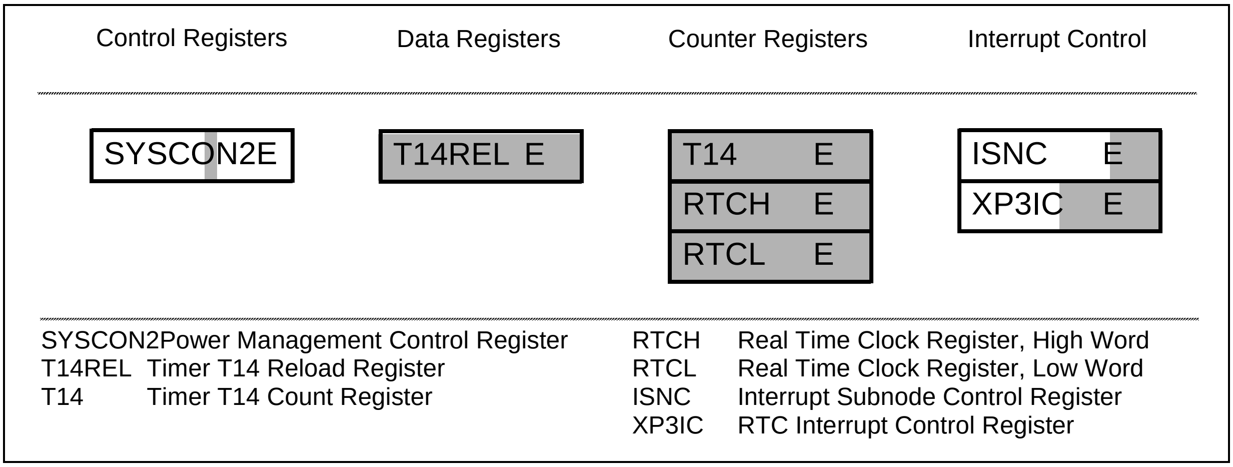
Watchdog Timer Reset is indicated after a reset triggered by an overflow of the watchdog timer.

Note: When bidirectional reset is enabled the $\overline{\text{RSTIN}}$ pin is pulled low for the duration of the internal reset sequence upon any sort of reset.
Therefore always a long hardware reset (LHWR) will be recognized in any case.

14 The Real Time Clock

The Real Time Clock (RTC) module of the C164 basically is an independent timer chain which is clocked directly with the oscillator clock and serves for different purposes:

- System clock to determine the current time and date
- Cyclic time based interrupt
- 48-bit timer for long term measurements



SYSCON2	Power Management Control Register	RTCH	Real Time Clock Register, High Word
T14REL	Timer T14 Reload Register	RTCL	Real Time Clock Register, Low Word
T14	Timer T14 Count Register	ISNC	Interrupt Subnode Control Register
		XP3IC	RTC Interrupt Control Register

Figure 14-1 SFRs Associated with the RTC Module

The RTC module consists of a chain of 3 divider blocks, a fixed 8:1 divider, the reloadable 16-bit timer T14 and the 32-bit RTC timer (accessible via registers RTCH and RTCL). Both timers count up.

The clock signal for the RTC module is directly derived from the on-chip oscillator frequency (not from the CPU clock) and fed through a separate clock driver. It is therefore independent from the selected clock generation mode of the C164 and is controlled by the clock generation circuitry.

Table 14-1 RTC Register Location within the ESFR space.

Register Name	Long/Short Address	Reset Value	Notes
T14	F0D2$_H$ / 69$_H$	UUUU$_H$	Prescaler timer, generates input clock for RTC register and periodic interrupt
T14REL	F0D0$_H$ / 68$_H$	UUUU$_H$	Timer reload register
RTCH	F0D6$_H$ / 6B$_H$	UUUU$_H$	High word of RTC register
RTCL	F0D4$_H$ / 6A$_H$	UUUU$_H$	Low word of RTC register

Note: The RTC registers are not affected by a reset. After a power on reset, however, they are undefined.

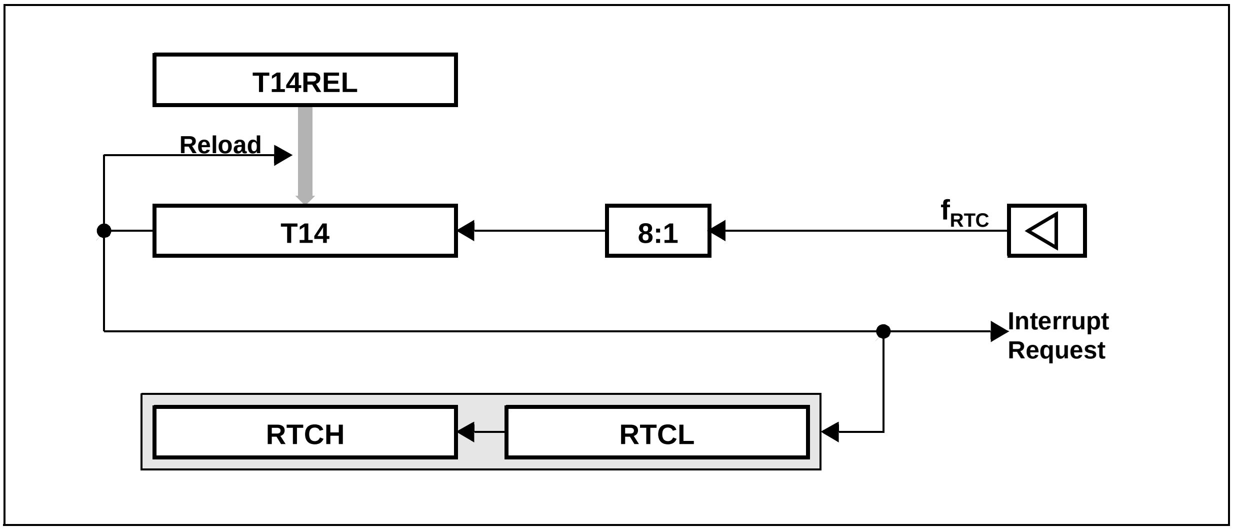


Figure 14-2 RTC Block Diagram

System Clock Operation

A real time system clock can be maintained that keeps on running also during idle mode and power down mode (optionally) and represents the current time and date. This is possible as the RTC module is not effected by a reset.
The maximum resolution (minimum stepwidth) for this clock information is determined by timer T14's input clock. The maximum usable timespan is achieved when T14REL is loaded with 0000_H and so T14 divides by 2^{16}.

Cyclic Interrupt Generation

The RTC module can generate an interrupt request whenever timer T14 overflows and is reloaded. This interrupt request may e.g. be used to provide a system time tick independent of the CPU frequency without loading the general purpose timers, or to wake up regularly from idle mode. The interrupt cycle time can be adjusted via the timer T14 reload register T14REL. Please refer to „RTC Interrupt Generation“ below for more details.

48-bit Timer Operation

The concatenation of the 16-bit reload timer T14 and the 32-bit RTC timer can be regarded as a 48-bit timer which is clocked with the RTC input frequency divided by the fixed prescaler. The reload register T14REL should be cleared to get a 48-bit binary timer. However, any other reload value may be used.
The maximum usable timespan is 2^{48} ($\approx 10^{14}$) T14 input clocks, which would equal more than 100 years at an oscillator frequency of 20 MHz.

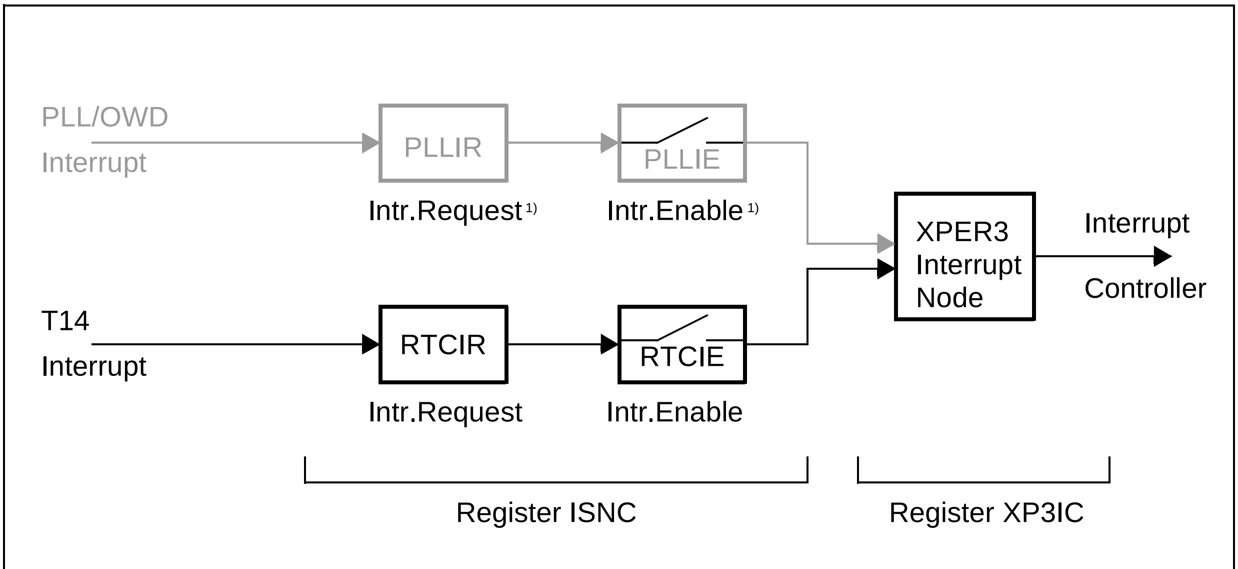
RTC Register Access

The actual value of the RTC is represented by the 3 registers T14, RTCL and RTCH. As these registers are concatenated to build the RTC counter chain, internal overflows occur while the RTC is running. When reading or writing the RTC value make sure to account for such internal overflows in order to avoid reading/writing corrupted values. When reading/writing e.g. 0000_H to RTCH and then accessing RTCL will produce a corrupted value as RTCL may overflow before it can be accessed. In this case, however, RTCH would be 0001_H. The same precautions must be taken for T14 and T14REL.

RTC Interrupt Generation

The RTC interrupt shares the XPER3 interrupt node with the PLL/OWD interrupt (if available). This is controlled by the interrupt subnode control register ISNC. The interrupt handler can determine the source of an interrupt request via the separate interrupt request and enable flags (see figure below) provided in register ISNC.

Note: If only one source is enabled no additional software check is required, of course.



Note: 1) Only available if PLL is implemented

Figure 14-3 RTC Interrupt Logic

If T14 interrupts are to be used both stages, the interrupt node (XP3IE='1') and the RTC subnode (RTCIE='1') must be enabled.

Please note that the node request bit XP3IR is automatically cleared when the interrupt handler is vectored to, while the subnode request bit RTCIR must be cleared by software.

Defining the RTC Time Base

The reload timer T14 determines the input frequency of the RTC timer, i.e. the RTC time base, as well as the T14 interrupt cycle time. The table below lists the interrupt period range and the T14 reload values (for a time base of 1 s and 1 ms) for several oscillator frequencies:

Table 14-2 RTC Interrupt Periods and Reload Values

Oscillator Frequency		RTC Interrupt Period		Reload Value A		Reload Value B	
		Minimum	Maximum	T14REL	Base	T14REL	Base
32.768KHz	Aux.	244.14 µs	16.0 s	$F000_H$	1.000 s	$FFFC_H$	0.977 ms
32 KHz	Aux.	250 µs	16.38 s	$F060_H$	1.000 s	$FFFC_H$	1.000 ms
32 KHz	Main	8000 µs	524.29 s	$FF83_H$	1.000 s	----	----
4 MHz	Main	64.0 µs	4.19 s	$C2F7_H$	1.000 s	$FFF0_H$	1.024 ms
5 MHz	Main	51.2 µs	3.35 s	$B3B5_H$	0.999 s	$FFEC_H$	1.024 ms
8 MHz	Main	32.0 µs	2.10 s	$85EE_H$	1.000 s	$FFE1_H$	0.992 ms
10 MHz	Main	25.6 µs	1.68 s	$676A_H$	0.999 s	$FFD9_H$	0.998 ms
12 MHz	Main	21.3 µs	1.40 s	$48E5_H$	1.000 s	$FFD2_H$	1.003 ms
16 MHz	Main	16.0 µs	1.05 s	$0BDC_H$	1.000 s	$FFC2_H$	0.992 ms

Increased RTC Accuracy through Software Correction

The accuracy of the C164's RTC is determined by the oscillator frequency and by the respective prescaling factor (excluding or including T14). The accuracy limit generated by the prescaler is due to the quantization of a binary counter (where the average is zero), while the accuracy limit generated by the oscillator frequency is due to the difference between ideal and real frequency (and therefore accumulates over time). The total accuracy of the RTC can be further increased via software for specific applications that demand a high time accuracy.

The key to the improved accuracy is the knowledge of the exact oscillator frequency. The relation of this frequency to the expected ideal frequency is a measure for the RTC's deviation. The number N of cycles after which this deviation causes an error of ±1 cycle can be easily computed. So the only action is to correct the count by ±1 after each series of N cycles.

This correction may be applied to the RTC register as well as to T14.

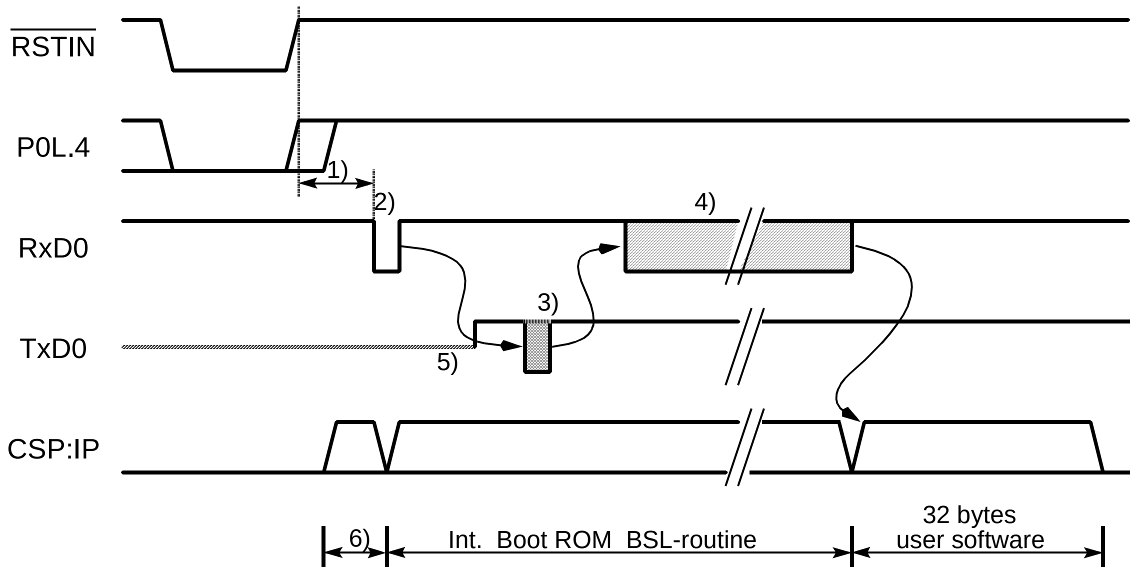
Also the correction may be done cyclic, e.g. within T14's interrupt service routine, or by evaluating a formula when the RTC registers are read (for this the respective „last“ RTC value must be available somewhere).

Note: For the majority of applications, however, the standard accuracy provided by the RTC's structure will be more than sufficient.

15 The Bootstrap Loader

The built-in bootstrap loader of the C164 provides a mechanism to load the startup program, which is executed after reset, via the serial interface. In this case no external memory or an internal ROM/OTP/Flash is required for the initialization code starting at location 00’0000$_H$.

The bootstrap loader moves code/data into the internal RAM, but it is also possible to transfer data via the serial interface into an external RAM using a second level loader routine. ROM memory (internal or external) is not necessary. However, it may be used to provide lookup tables or may provide “core-code”, i.e. a set of general purpose subroutines, e.g. for IO operations, number crunching, system initialization, etc.



1) BSL initialization time, < 70/f_{CPU} µs, (f_{CPU} in [MHz]).
2) Zero byte (1 start bit, eight ‘0’ data bits, 1 stop bit), sent by host.
3) Identification byte, sent by C164.
4) 32 bytes of code / data, sent by host.
5) Caution: TxD0 is only driven a certain time after reception of the zero byte (< 40/f_{CPU} µs, f_{CPU} in [MHz]).
6) Internal Boot ROM.

Figure 15-1 Bootstrap Loader Sequence

The Bootstrap Loader may be used to load the complete application software into ROMless systems, it may load temporary software into complete systems for testing or calibration, it may also be used to load a programming routine for Flash devices.

The BSL mechanism may be used for standard system startup as well as only for special occasions like system maintenance (firmware update) or end-of-line programming or testing.

Entering the Bootstrap Loader

The C164 enters BSL mode if pin P0L.4 is sampled low at the end of a hardware reset. In this case the built-in bootstrap loader is activated independent of the selected bus mode. The bootstrap loader code is stored in a special Boot-ROM, no part of the standard mask ROM, OTP or Flash memory area is required for this.

After entering BSL mode and the respective initialization[1)] the C164 scans the RXD0 line to receive a zero byte, i.e. one start bit, eight '0' data bits and one stop bit. From the duration of this zero byte it calculates the corresponding baudrate factor with respect to the current CPU clock, initializes the serial interface ASC0 accordingly and switches pin TxD0 to output. Using this baudrate, an identification byte is returned to the host that provides the loaded data.

This identification byte identifies the device to be bootet. The following codes are defined:

55_H:	8xC166.
$A5_H$:	Previous versions of the C167 (obsolete).
$B5_H$:	Previous versions of the C165.
$C5_H$:	C167 derivatives.
$D5_H$:	All devices equipped with identification registers.

Note: The identification byte $D5_H$ does not directly identify a specific derivative. This information can in this case be obtained from the identification registers.

When the C164 has entered BSL mode, the following configuration is automatically set (values that deviate from the normal reset values, are ***marked***):

Watchdog Timer:	***Disabled***	Register STKUN:	$FA40_H$
Context Pointer CP:	$FA00_H$	Register STKOV:	$FA0C_H$ 0<->C
Stack Pointer SP:	$FA40_H$	Register BUSCON0:	acc. to startup config.
Register S0CON:	***8011_H***	P3.10 / TXD0:	'***1***'
Register S0BG:	acc. to '00' byte	DP3.10:	'***1***'

Other than after a normal reset the watchdog timer is disabled, so the bootstrap loading sequence is not time limited. Pin TXD0 is configured as output, so the C164 can return the identification byte.

Note: Even if the internal ROM/OTP/Flash is enabled, no code can be executed out of it.

The hardware that activates the BSL during reset may be a simple pull-down resistor on P0L.4 for systems that use this feature upon every hardware reset. You may want to use

[1)] The external host should not send the zero byte before the end of the BSL initialization time (see figure) to make sure that it is correctly received.

a switchable solution (via jumper or an external signal) for systems that only temporarily use the bootstrap loader.

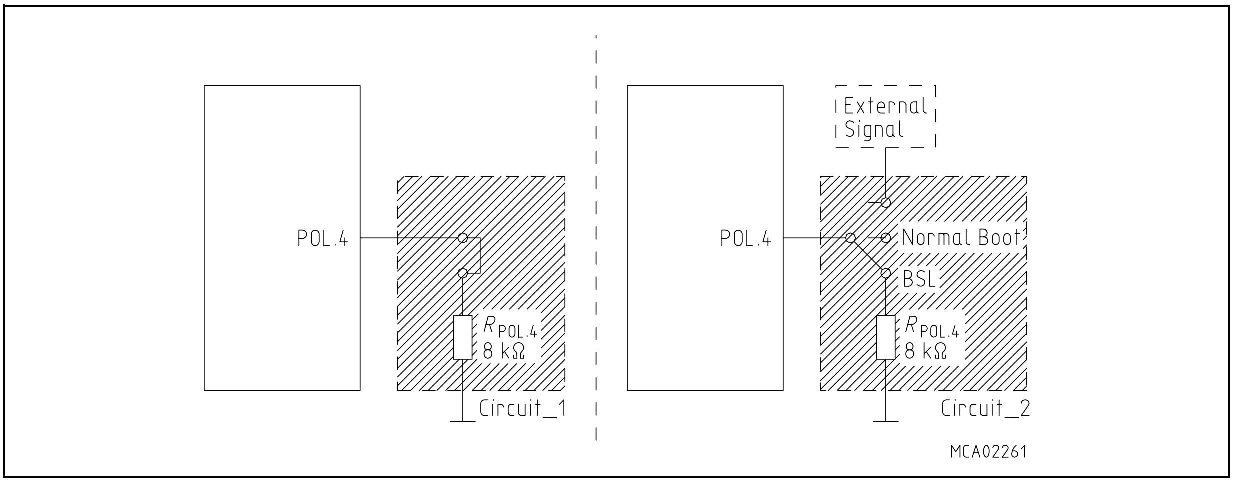


Figure 15-2 Hardware Provisions to Activate the BSL

After sending the identification byte the ASC0 receiver is enabled and is ready to receive the initial 32 bytes from the host. A half duplex connection is therefore sufficient to feed the BSL.

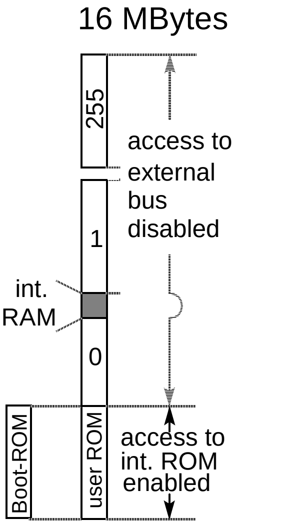
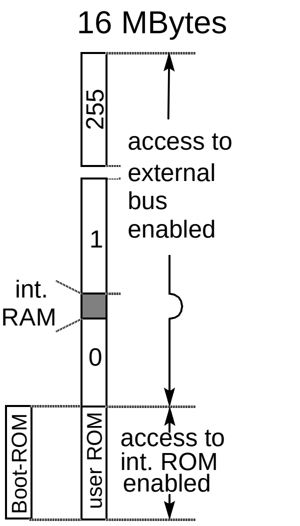
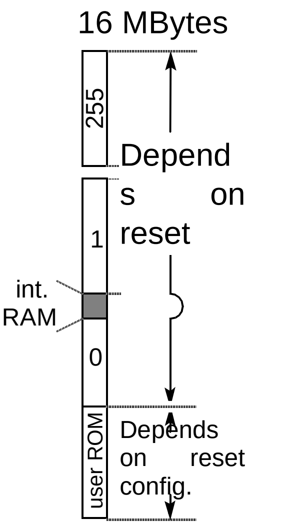
Note: In order to properly enter BSL mode it is not only required to pull P0L.4 low, but also pins P0L.2, P0L.3, P0L.5 must receive defined levels. This is described in chapter „System Reset“.

Memory Configuration after Reset

The configuration (i.e. the accessibility) of the C164's memory areas after reset in Bootstrap-Loader mode differs from the standard case. Pin $\overline{\text{EA}}$ is not evaluated when BSL mode is selected, and accesses to the internal code memory are partly redirected, while the C164 is in BSL mode (see table below). All code fetches are made from the special Boot-ROM, while data accesses read from the internal code memory. Data accesses will return undefined values on ROMless devices.

Note: The code in the Boot-ROM is not an invariant feature of the C164. User software should not try to execute code from the internal ROM area while the BSL mode is still active, as these fetches will be redirected to the Boot-ROM.
The Boot-ROM will also "move" to segment 1, when the internal ROM area is mapped to segment 1.

Table 15-1 BSL Memory Configurations

	 16 MBytes 255 1 0 int. RAM access to external bus disabled Boot-ROM user ROM access to int. ROM enabled	 16 MBytes 255 1 0 int. RAM access to external bus enabled Boot-ROM user ROM access to int. ROM enabled	 16 MBytes 255 1 0 int. RAM Depends on reset user ROM Depends on reset config.
BSL mode active	**Yes** (P0L.4='0')	**Yes** (P0L.4='0')	**No** (P0L.4='1')
$\overline{\text{EA}}$ pin	high	low	acc. to application
Code fetch from internal ROM area	Boot-ROM access	Boot-ROM access	User ROM access
Data fetch from internal ROM area	User ROM access	User ROM access	User ROM access

Loading the Startup Code

After sending the identification byte the BSL enters a loop to receive 32 bytes via ASC0. These bytes are stored sequentially into locations 00’FA40$_H$ through 00’FA5F$_H$ of the internal RAM. So up to 16 instructions may be placed into the RAM area. To execute the loaded code the BSL then jumps to location 00’FA40$_H$, i.e. the first loaded instruction. The bootstrap loading sequence is now terminated, the C164 remains in BSL mode, however. Most probably the initially loaded routine will load additional code or data, as an average application is likely to require substantially more than 16 instructions. This second receive loop may directly use the pre-initialized interface ASC0 to receive data and store it to arbitrary user-defined locations.

This second level of loaded code may be the final application code. It may also be another, more sophisticated, loader routine that adds a transmission protocol to enhance the integrity of the loaded code or data. It may also contain a code sequence to change the system configuration and enable the bus interface to store the received data into external memory.

This process may go through several iterations or may directly execute the final application. In all cases the C164 will still run in BSL mode, i.e. with the watchdog timer disabled and limited access to the internal code memory. All code fetches from the internal ROM area (00’0000$_H$...00’7FFF$_H$ or 01’0000$_H$...01’7FFF$_H$, if mapped to segment 1) are redirected to the special Boot-ROM. Data fetches access will access the internal code memory of the C164, if any is available, but will return undefined data on ROMless devices.

Exiting Bootstrap Loader Mode

In order to execute a program in normal mode, the BSL mode must be terminated first. The C164 exits BSL mode upon a software reset (ignores the level on P0L.4) or a hardware reset (P0L.4 must be high then!). After a reset the C164 will start executing from location 00’0000$_H$ of the internal ROM or the external memory, as programmed via pin $\overline{\text{EA}}$.

Choosing the Baudrate for the BSL

The calculation of the serial baudrate for ASC0 from the length of the first zero byte that is received, allows the operation of the bootstrap loader of the C164 with a wide range of baudrates. However, the upper and lower limits have to be kept, in order to insure proper data transfer.

$$B_{C164} = \frac{f_{CPU}}{32 \cdot (S0BRL + 1)}$$

The C164 uses timer T3 to measure the length of the initial zero byte. The quantization uncertainty of this measurement implies the first deviation from the real baudrate, the next deviation is implied by the computation of the S0BRL reload value from the timer contents. The formula below shows the association:

$$S0BRL = \frac{T3 - 18}{36} \quad , \qquad T3 = \frac{9}{8} \cdot \frac{f_{CPU}}{B_{Host}}$$

For a correct data transfer from the host to the C164 the maximum deviation between the internal initialized baudrate for ASC0 and the real baudrate of the host should be below 2.5%. The deviation (F_B, in percent) between host baudrate and C164 baudrate can be calculated via the formula below:

$$F_B = \left| \frac{B_{Contr} - B_{Host}}{B_{Contr}} \right| \cdot 100 \ \% \quad , \qquad F_B \leq 2{,}5 \ \%$$

*Note: **F**unction (F_B) does not consider the tolerances of oscillators and other devices supporting the serial communication.*

This baudrate deviation is a nonlinear function depending on the CPU clock and the baudrate of the host. The maxima of the function (F_B) increase with the host baudrate due to the smaller baudrate prescaler factors and the implied higher quantization error (see figure below).

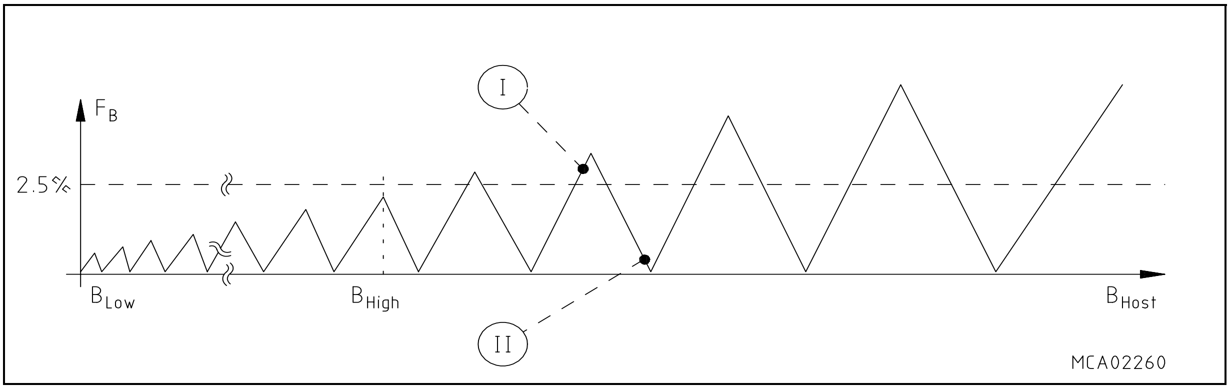


Figure 15-3 Baudrate Deviation Between Host and C164

The minimum baudrate (B_{Low} in the figure above) is determined by the maximum count capacity of timer T3, when measuring the zero byte, i.e. it depends on the CPU clock. The minimum baudrate is obtained by using the maximum T3 count 2^{16} in the baudrate formula. Baudrates below B_{Low} would cause T3 to overflow. In this case ASC0 cannot be initialized properly and the communication with the external host is likely to fail.

The maximum baudrate (B_{High} in the figure above) is the highest baudrate where the deviation still does not exceed the limit, i.e. all baudrates between B_{Low} and B_{High} are below the deviation limit. B_{High} marks the baudrate up to which communication with the external host will work properly without additional tests or investigations.

Higher baudrates, however, may be used as long as the actual deviation does not exceed the indicated limit. A certain baudrate (marked I) in the figure) may e.g. violate the deviation limit, while an even higher baudrate (marked II) in the figure) stays very well below it. Any baudrate can be used for the bootstrap loader provided that the following three prerequisites are fulfilled:

- the baudrate is within the specified operating range for the ASC0
- the external host is able to use this baudrate
- the computed deviation error is below the limit.

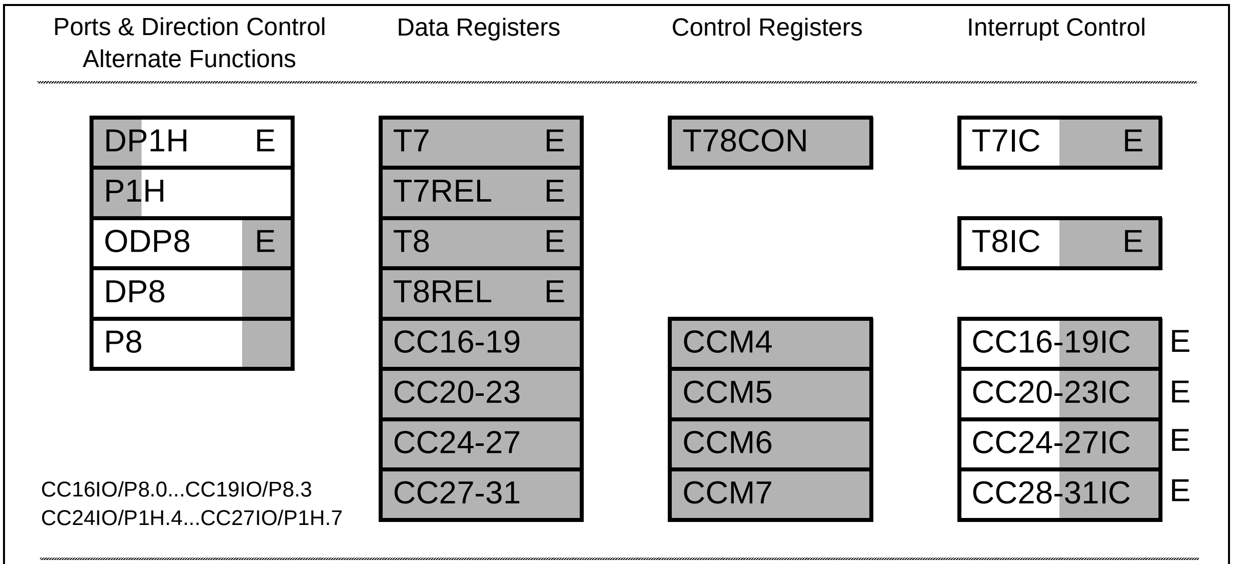
Table 15-2 Bootstrap Loader Baudrate Ranges

f_{CPU} [MHz]	**10**	**12**	**16**	**20**	**25**
$\mathbf{B_{MAX}}$	312,500	375,000	500,000	625,000	781,250
$\mathbf{B_{High}}$	9,600	19,200	19,200	19,200	38,400
$\mathbf{B_{STDmin}}$	600	600	600	600	600
$\mathbf{B_{Low}}$	172	206	275	343	429

16 The Capture / Compare Unit

The C164 provides a Capture/Compare (CAPCOM) unit which provides 16 channels (8 IO pins) which interact with 2 timers. The CAPCOM units can **capture** the contents of a timer on specific internal or external events, or can **compare** a timer's content with given values and modify output signals in case of a match. With this mechanism it supports generation and control of timing sequences on up to 16 channels with a minimum of software intervention.

From the programmer's point of view, the term 'CAPCOM unit' refers to a set of SFRs which are associated with this peripheral, including the port pins which may be used for alternate input/output functions including their direction control bits.



ODP8	Port 8 Open Drain Control Register	CCM4	CAPCOM2 Mode Control Register 4
DPx	Port x Direction Control Register	CCM5	CAPCOM2 Mode Control Register 5
Px	Port x Data Register	CCM6	CAPCOM2 Mode Control Register 6
		CCM7	CAPCOM2 Mode Control Register 7
TxREL	CAPCOM2 Timer x Reload Register	T78CON	CAPCOM2 Timers T7 and T8 Control Register
Tx	CAPCOM2 Timer x Register	TxIC	CAPCOM2 Timer x Interrupt Control Register
CC16...19	CAPCOM2 Register 16...19	CC16...19IC	CAPCOM2 Interrupt Control Register 16...19
CC20...23	CAPCOM2 Register 20...23	CC20...23IC	CAPCOM2 Interrupt Control Register 20...23
CC24...27	CAPCOM2 Register 24...27	CC24...27IC	CAPCOM2 Interrupt Control Register 24...27
CC28...31	CAPCOM2 Register 28...31	CC28...31IC	CAPCOM2 Interrupt Control Register 28...31

Figure 16-1 SFRs and Port Pins associated with the CAPCOM Unit

The CAPCOM2 unit is typically used to handle high speed IO tasks such as pulse and waveform generation, pulse width modulation, or recording of the time at which specific events occur. It also allows the implementation of up to 16 software timers. The maximum resolution of the CAPCOM2 unit is 8 CPU clock cycles (=16 TCL).

The CAPCOM2 unit consists of two 16-bit timers (T7 / T8), each with its own reload register (TxREL), and a bank of 16 dual purpose 16-bit capture/compare registers (CC16 through CC31).

The input clock for the CAPCOM timers is programmable to several prescaled values of the CPU clock, or it can be derived from an overflow/underflow of timer T3 in block GPT1. T7 may also operate in counter mode (from an external input) where it can be clocked by external events.

Each capture/compare register may be programmed individually for capture or compare function, and each register may be allocated to either timer. Eight capture/compare registers have one port pin associated with it, respectively, which serves as an input pin for the capture function or as an output pin for the compare function. The capture function causes the current timer contents to be latched into the respective capture/compare register triggered by an event (transition) on its associated port pin. The compare function may cause an output signal transition on that port pin whose associated capture/compare register matches the current timer contents. Specific interrupt requests are generated upon each capture/compare event or upon timer overflow.

The figure below shows the basic structure of the two CAPCOM units.

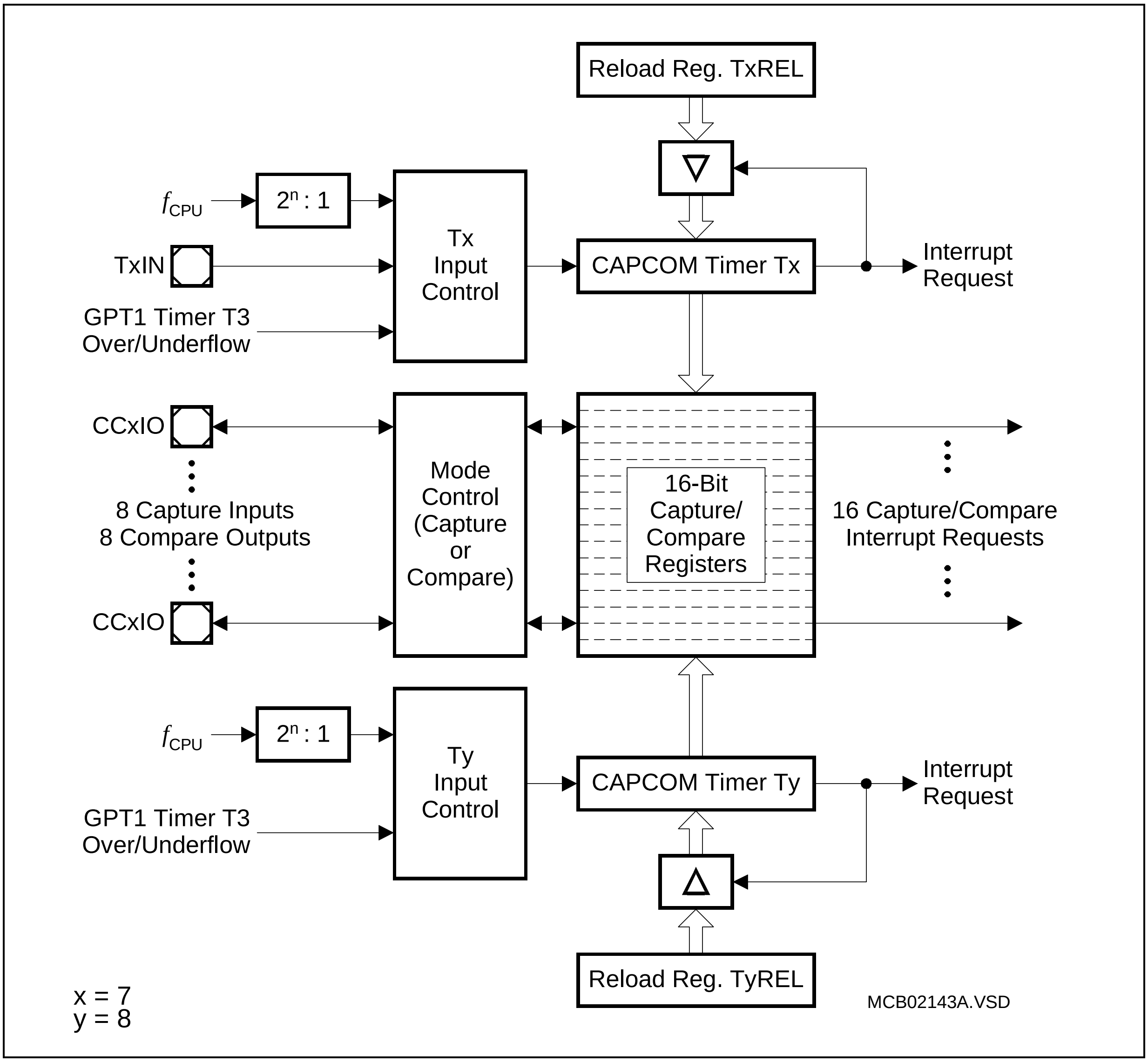


Figure 16-2 CAPCOM Unit Block Diagram

Table 16-1 CAPCOM Channel Port Connections

Unit	Channel	Port	Capture	Compare
CAPCOM2	CC31IO...CC28IO	-	-	-
	CC27IO...CC24IO	P1H.7...P1H.4	Input	Output
	CC23IO...CC20IO	-	-	-
	CC19IO...CC16IO	P8.3...P8.0	Input	Output
	Σ = 16	Σ = 8	Σ = 8	Σ = 8

16.1 The CAPCOM Timers

The primary use of the timers T7/T8 is to provide two independent time bases (16 TCL maximum resolution) for the capture/compare registers of each unit, but they may also be used independent of the capture/compare registers.
The basic structure of the two timers is identical, while the selection of input signals is different for timer T7 and timer T8 (see figures below).

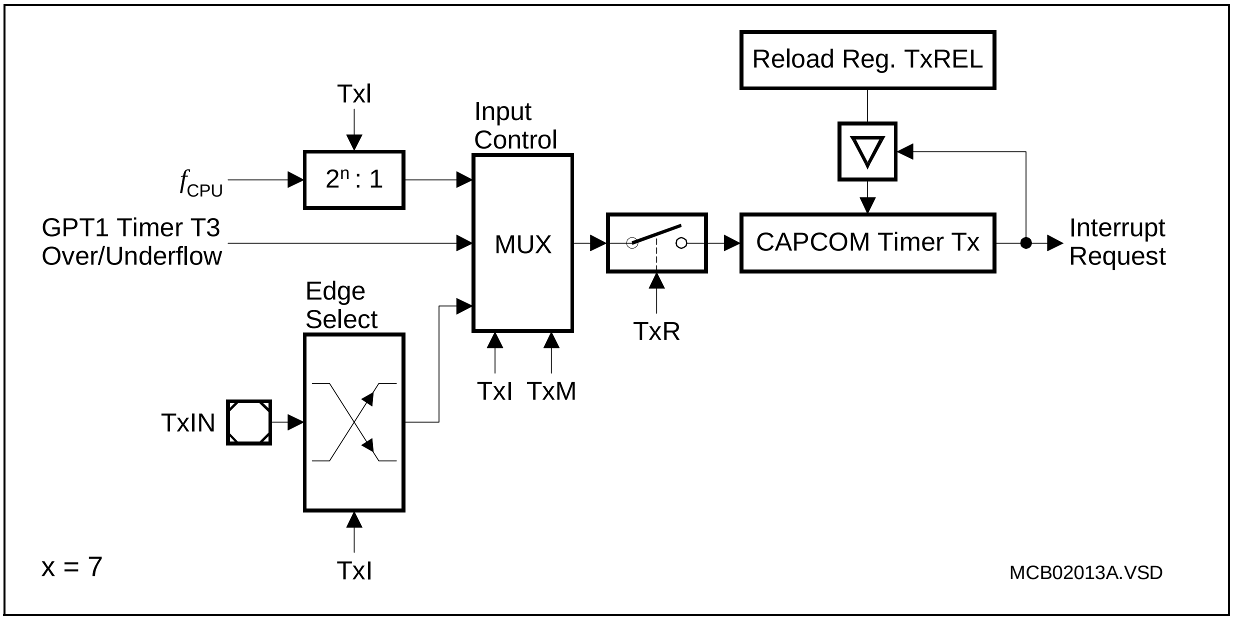


Figure 16-3 Block Diagram of CAPCOM Timer T7

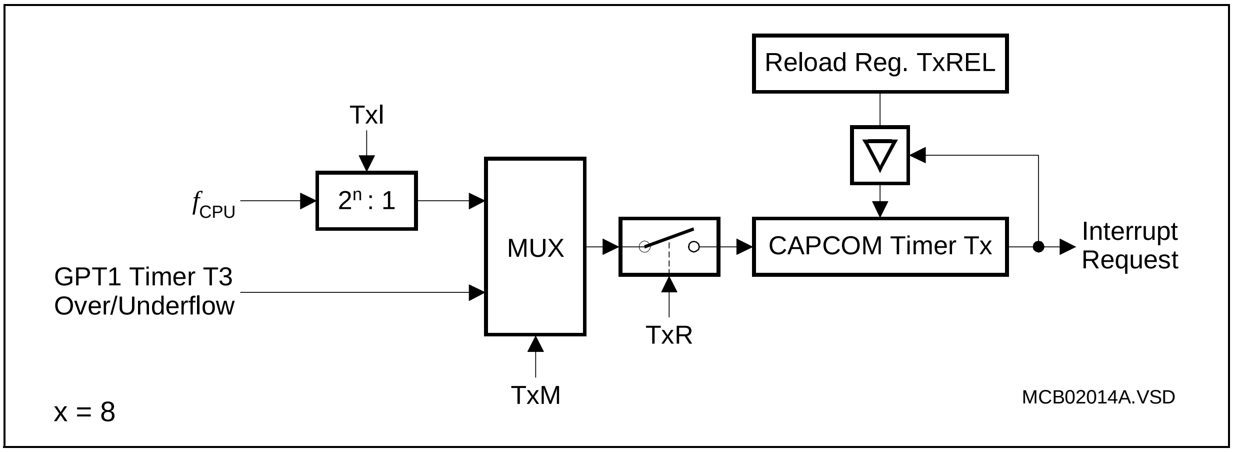


Figure 16-4 Block Diagram of CAPCOM Timer T8

The functions of the CAPCOM timers are controlled via the bitaddressable 16-bit control register T78CON. The high-byte of T78CON controls T8, the low-byte of T78CON controls T7. The control options are identical for both timers (except for external input).

T78CON
CAPCOM Timer 7/8 Ctrl. Reg. **SFR (FF20$_H$/90$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**T8R**	**-**	**-**	**T8M**	**T8I**			**-**	**T7R**	**-**	**-**	**T7M**	**T7I**		
-	rw	-	-	rw	rw			-	rw	-	-	rw	rw		

Bit	Function
TxI	**Timer / Counter x Input Selection** Timer Mode (TxM='0') Input Frequency = f_{CPU} / $2^{(<TxI>+3)}$ See also table below for examples. Counter Mode (TxM='1'):000 Overflow/Underflow of GPT1 Timer 3 001 Positive (rising) edge on pin T7IN *) 010 Negative (falling) edge on pin T7IN *) 011 Any edge (rising and falling) on pin T7IN *) 1XX Reserved
TxM	**Timer / Counter x Mode Selection** 0: Timer Mode (Input derived from internal clock) 1: Counter Mode (Input from External Input or T3)
TxR	**Timer / Counter x Run Control** 0: Timer/Counter x is disabled 1: Timer/Counter x is enabled

*) This selection is available for timer T7. Timer T8 will stop at this selection!

The timer run flags T7R, and T8R allow for enabling and disabling the timers. The following description of the timer modes and operation always applies to the enabled state of the timers, i.e. the respective run flag is assumed to be set to '1'.

In all modes, the timers are always counting upward. The current timer values are accessible for the CPU in the timer registers Tx, which are non-bitaddressable SFRs. When the CPU writes to a register Tx in the state immediately before the respective timer increment or reload is to be performed, the CPU write operation has priority and the increment or reload is disabled to guarantee correct timer operation.

Timer Mode

The bits TxM in SFRs T78CON select between timer or counter mode for the respective timer. In timer mode (TxM='0'), the input clock for a timer is derived from the internal CPU clock divided by a programmable prescaler. The different options for the prescaler are selected separately for each timer by the bit fields TxI.

The input frequencies f_{Tx} for Tx are determined as a function of the CPU clock as follows, where <TxI> represents the contents of the bit field TxI:

$$f_{Tx} = \frac{f_{CPU}}{2^{(<TxI>+3)}}$$

When a timer overflows from $FFFF_H$ to 0000_H it is reloaded with the value stored in its respective reload register TxREL. The reload value determines the period P_{Tx} between two consecutive overflows of Tx as follows:

$$P_{Tx} = \frac{(2^{16} - <TxREL>) * 2^{(<TxI>+3)}}{f_{CPU}}$$

The timer input frequencies, resolution and periods which result from the selected prescaler option in TxI when using a 25 MHz CPU clock are listed in the table below. The numbers for the timer periods are based on a reload value of 0000_H. Note that some numbers may be rounded to 3 significant digits.

Table 16-2 Timer Input Frequencies, Resolution and Period

f_{CPU} = 25 MHz	Timer Input Selection TxI							
	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B
Prescaler for f_{CPU}	8	16	32	64	128	256	512	1024
Input Frequency	3.125 MHz	1.563 MHz	781.25 KHz	390.63 KHz	195.31 KHz	97.656 KHz	48.828 KHz	24.414 KHz
Resolution	320 ns	640 ns	1.28 µs	2.56 µs	5.12 µs	10.24 µs	20.48 µs	40.96 µs
Period	21 ms	42 ms	84 ms	168 ms	336 ms	672 ms	1.344 s	2.688 s

After a timer has been started by setting its run flag (TxR) to '1', the first increment will occur within the time interval which is defined by the selected timer resolution. All further increments occur exactly after the time defined by the timer resolution.

When both timers of a CAPCOM unit are to be incremented or reloaded at the same time T7 is always serviced one CPU clock before T8.

Counter Mode

The bits TxM in SFR T78CON select between timer or counter mode for the respective timer. In Counter mode (TxM=‘1’) the input clock for a timer can be derived from the overflows/underflows of timer T3 in block GPT1. In addition, timer T7 can be clocked by external events. Either a positive, a negative, or both a positive and a negative transition at pin T7IN (alternate port input function), respectively, can be selected to cause an increment of T7.

When T8 is programmed to run in counter mode, bit field TxI is used to enable the overflows/underflows of timer T3 as the count source. This is the only option for T8 and it is selected by the combination TxI=000_B. When bit field TxI is programmed to any other valid combination, the respective timer will stop.

When T7 is programmed to run in counter mode, bit field TxI is used to select the count source and transition (if the source is the input pin) which should cause a count trigger (see description of TxyCON for the possible selections).

Note: In order to use pin T7IN as external count input pin, the respective port pin must be configured as input, i.e. the corresponding direction control bit must be cleared (DPx.y=’0’).
If the respective port pin is configured as output, the associated timer may be clocked by modifying the port output latches Px.y via software, e.g. for testing purposes.

The maximum external input frequency to T7 in counter mode is f_{CPU}/16. To ensure that a signal transition is properly recognized at the timer input, an external count input signal should be held for at least 8 CPU clock cycles before it changes its level again. The incremented count value appears in SFR T7 within 8 CPU clock cycles after the signal transition at pin T7IN.

Reload

A reload of a timer with the 16-bit value stored in its associated reload register in both modes is performed each time a timer would overflow from $FFFF_H$ to 0000_H. In this case the timer does not wrap around to 0000_H, but rather is reloaded with the contents of the respective reload register TxREL. The timer then resumes incrementing starting from the reloaded value.

The reload registers TxREL are not bitaddressable.

16.2 CAPCOM Unit Timer Interrupts

Upon a timer overflow the corresponding timer interrupt request flag TxIR for the respective timer will be set. This flag can be used to generate an interrupt or trigger a PEC service request, when enabled by the respective interrupt enable bit TxIE.

Each timer has its own bitaddressable interrupt control register (TxIC) and its own interrupt vector (TxINT). The organization of the interrupt control registers TxIC is identical with the other interrupt control registers.

T7IC
CAPCOM T7 Intr. Ctrl. Reg. **ESFR (F17A$_H$/BE$_H$)** **Reset value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T7IR	**T7IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

T8IC
CAPCOM T8 Intr. Ctrl. Reg. **ESFR (F17C$_H$/BF$_H$)** **Reset value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T8IR	**T8IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

16.3 Capture/Compare Registers

The 16-bit capture/compare registers CC16 through CC27 are used as data registers for capture or compare operations with respect to timers T7/T8. The capture/compare registers are not bitaddressable.

Each of the registers CCx may be individually programmed for capture mode or one of 4 different compare modes, and may be allocated individually to one of the two timers T7 or T8, respectively. A special combination of compare modes additionally allows the implementation of a 'double-register' compare mode. When capture or compare operation is disabled for one of the CCx registers, it may be used for general purpose variable storage.

Capture/Compare Mode Registers for the CAPCOM2 Unit

The functions of the 16 capture/compare registers are controlled by 4 bitaddressable 16-bit mode control registers named CCM4...CCM7 which are organized identically (see description below). Each register contains bits for mode selection and timer allocation of four capture/compare registers.

CCM4
CAPCOM Mode Ctrl. Reg. 4 SFR (FF22$_H$/91$_H$) Reset value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 19	CCMOD19			ACC 18	CCMOD18			ACC 17	CCMOD17			ACC 16	CCMOD16		
rw	rw			rw	rw			rw	rw			rw	rw		

CCM5
CAPCOM Mode Ctrl. Reg. 5 SFR (FF24$_H$/92$_H$) Reset value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 23	CCMOD23			ACC 22	CCMOD22			ACC 21	CCMOD21			ACC 20	CCMOD20		
rw	rw			rw	rw			rw	rw			rw	rw		

CCM6
CAPCOM Mode Ctrl. Reg. 6 SFR (FF26$_H$/93$_H$) Reset value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 27	CCMOD27			ACC 26	CCMOD26			ACC 25	CCMOD25			ACC 24	CCMOD24		
rw	rw			rw	rw			rw	rw			rw	rw		

CCM7
CAPCOM Mode Ctrl. Reg. 7 SFR ($FF28_H/94_H$) Reset value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 31	CCMOD31			ACC 30	CCMOD30			ACC 29	CCMOD29			ACC 28	CCMOD28		
rw	rw			rw	rw			rw	rw			rw	rw		

Bit	Function
CCMODx	**Mode Selection for Capture/Compare Register CCx** The available capture/compare modes are listed in the table below.
ACCx	**Allocation Bit for Capture/Compare Register CCx** 0: CCx allocated to Timer T7 1: CCx allocated to Timer T8

Table 16-3 Selection of Capture Modes and Compare Modes

CCMODx	Selected Operating Mode
0 0 0	**Disable Capture and Compare Modes** The respective CAPCOM register may be used for general variable storage.
0 0 1	**Capture on Positive Transition (Rising Edge) at Pin CCxIO**
0 1 0	**Capture on Negative Transition (Falling Edge) at Pin CCxIO**
0 1 1	**Capture on Positive and Negative Transition (Both Edges) at Pin CCxIO**
1 0 0	**Compare Mode 0: Interrupt Only** Several interrupts per timer period. Enables double-register compare mode for registers CC24...CC27.
1 0 1	**Compare Mode 1: Toggle Output Pin on each Match** Several compare events per timer period. This mode is required for double-register compare mode for registers CC16...CC19.
1 1 0	**Compare Mode 2: Interrupt Only** Only one interrupt per timer period.
1 1 1	**Compare Mode 3: Set Output Pin on each Match** Reset output pin on each timer overflow; Only one interrupt per timer period.

The detailed discussion of the capture and compare modes is valid for all the capture/compare channels, so registers, bits and pins are only referenced by the placeholder ‘x’.

Note: Only capture/compare channels 16...19 and 24...27 are connected to pins.
A capture or compare event on channel 27 may be used to trigger a channel injection on the C164’s A/D converter if enabled.

16.4 Capture Mode

In response to an external event the content of the associated timer (T7 or T8, depending on the state of the allocation control bit ACCx) is latched into the respective capture register CCx . The external event causing a capture can be programmed to be either a positive, a negative, or both a positive or a negative transition at the respective external input pin CCxIO.

The triggering transition is selected by the mode bits CCMODx in the respective CAPCOM mode control register. In any case, the event causing a capture will also set the respective interrupt request flag CCxIR, which can cause an interrupt or a PEC service request, when enabled.

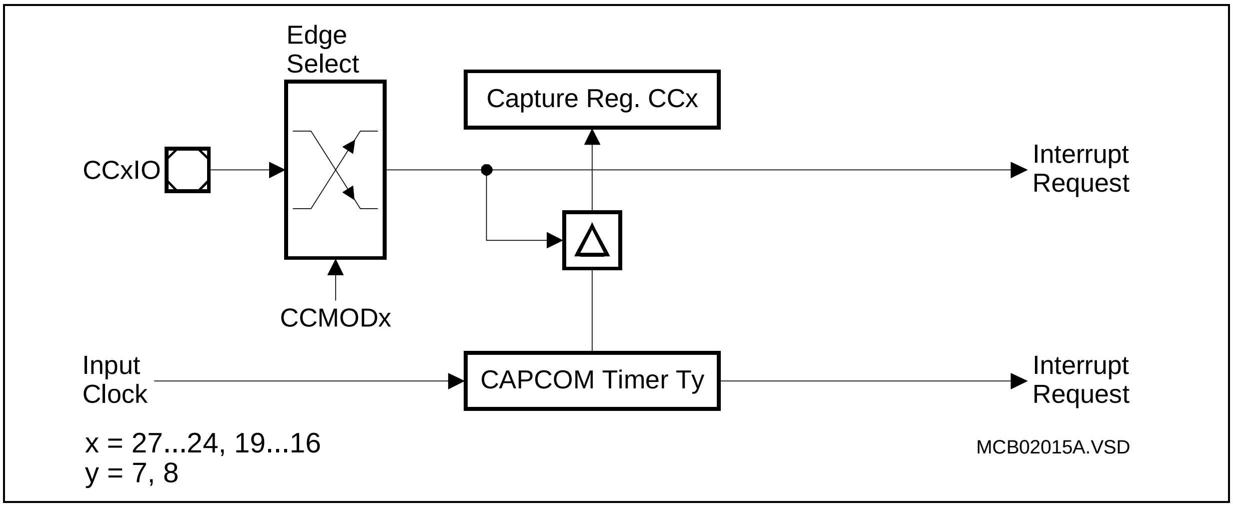


Figure 16-5 Capture Mode Block Diagram

In order to use the respective port pin as external capture input pin CCxIO for capture register CCx, this port pin must be configured as input, i.e. the corresponding direction control bit must be set to ‘0’. To ensure that a signal transition is properly recognized, an external capture input signal should be held for at least 8 CPU clock cycles before it changes its level.

During these 8 CPU clock cycles the capture input signals are scanned sequentially. When a timer is modified or incremented during this process, the new timer contents will already be captured for the remaining capture registers within the current scanning sequence.

If pin CCxIO is configured as output, the capture function may be triggered by modifying the corresponding port output latch via software, e.g. for testing purposes.

16.5 Compare Modes

The compare modes allow triggering of events (interrupts and/or output signal transitions) with minimum software overhead. In all compare modes, the 16-bit value stored in compare register CCx (in the following also referred to as ’compare value’) is continuously compared with the contents of the allocated timer (T7 or T8). If the current timer contents match the compare value, an appropriate output signal, which is based on the selected compare mode, can be generated at the corresponding output pin CCxIO and the associated interrupt request flag CCxIR is set, which can generate an interrupt request (if enabled).

As for capture mode, the compare registers are also processed sequentially during compare mode. When any two compare registers are programmed to the same compare value, their corresponding interrupt request flags will be set to ’1’ and the selected output signals will be generated within 8 CPU clock cycles after the allocated timer is incremented to the compare value. Further compare events on the same compare value are disabled until the timer is incremented again or written to by software. After a reset, compare events for register CCx will only become enabled, if the allocated timer has been incremented or written to by software and one of the compare modes described in the following has been selected for this register.

The different compare modes which can be programmed for a given compare register CCx are selected by the mode control field CCMODx in the associated capture/compare mode control register. In the following, each of the compare modes including the special ’double register’ mode is discussed in detail.

Compare Mode 0

This is an interrupt-only mode which can be used for software timing purposes. Compare mode 0 is selected for a given compare register CCx by setting bit field CCMODx of the corresponding mode control register to ‘100_B’.

In this mode, the interrupt request flag CCxIR is set each time a match is detected between the content of compare register CCx and the allocated timer. Several of these compare events are possible within a single timer period, when the compare value in register CCx is updated during the timer period. The corresponding port pin CCxIO is not affected by compare events in this mode and can be used as general purpose IO pin.

If compare mode 0 is programmed for one of the registers CC24...CC27, the double-register compare mode becomes enabled for this register if the corresponding bank 1 register is programmed to compare mode 1 (see section “Double-Register Compare Mode”).

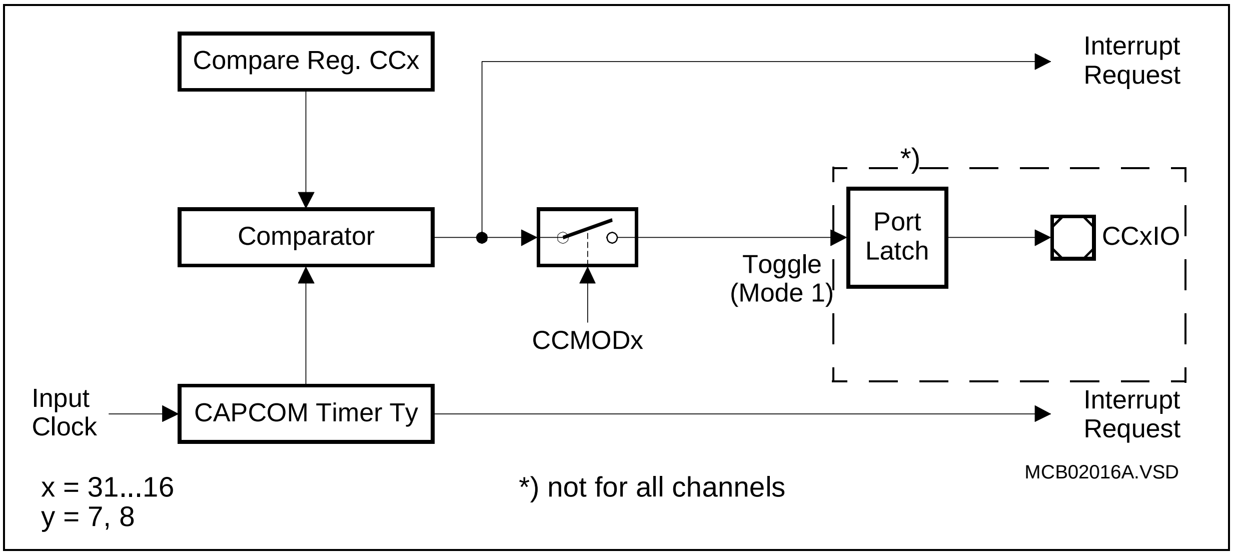


Figure 16-6 Compare Mode 0 and 1 Block Diagram

Note: The port latch and pin remain unaffected in compare mode 0.

In the example below, the compare value in register CCx is modified from cv1 to cv2 after compare events #1 and #3, and from cv2 to cv1 after events #2 and #4, etc. This results in periodic interrupt requests from timer Ty, and in interrupt requests from register CCx which occur at the time specified by the user through cv1 and cv2.

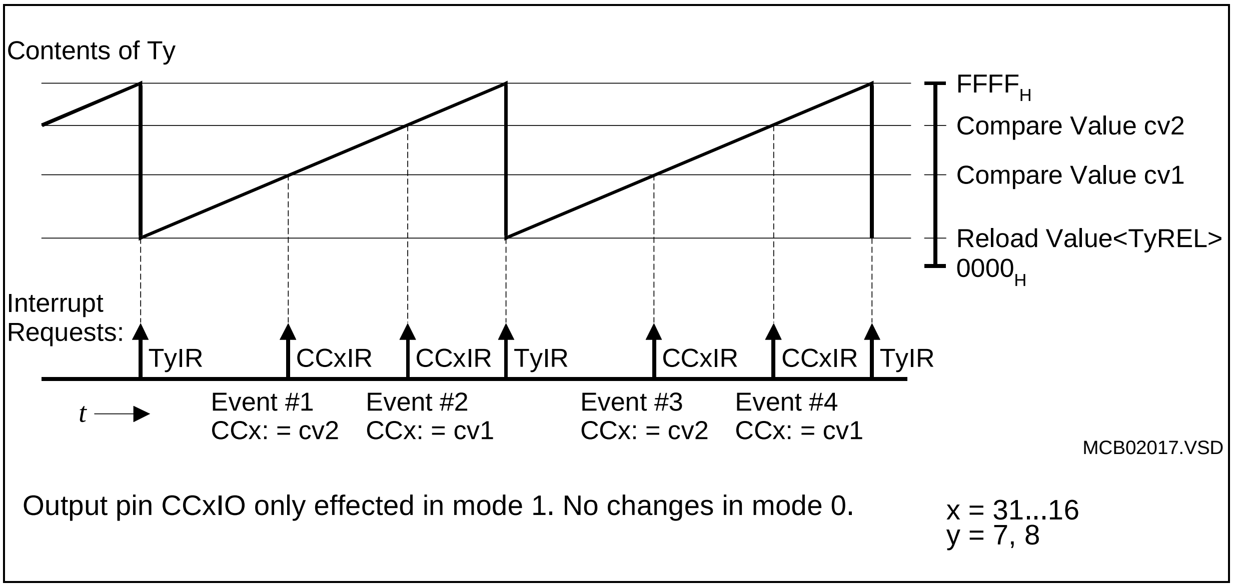


Figure 16-7 Timing Example for Compare Modes 0 and 1

Compare Mode 1

Compare mode 1 is selected for register CCx by setting bit field CCMODx of the corresponding mode control register to ‘101_B’.

When a match between the content of the allocated timer and the compare value in register CCx is detected in this mode, interrupt request flag CCxIR is set to ‘1’, and in addition the corresponding output pin CCxIO (alternate port output function) is toggled. For this purpose, the state of the respective port output latch (not the pin) is read, inverted, and then written back to the output latch.

Compare mode 1 allows several compare events within a single timer period. An overflow of the allocated timer has no effect on the output pin, nor does it disable or enable further compare events.

In order to use the respective port pin as compare signal output pin CCxIO for compare register CCx in compare mode 1, this port pin must be configured as output, i.e. the corresponding direction control bit must be set to ‘1’. With this configuration, the initial state of the output signal can be programmed or its state can be modified at any time by writing to the port output latch.

In compare mode 1 the port latch is toggled upon each compare event (see Timing Example above).

If compare mode 1 is programmed for one of the registers CC16...CC19 the double-register compare mode becomes enabled for this register if the corresponding bank 2 register is programmed to compare mode 0 (see section “Double-Register compare Mode”).

Note: If the port output latch is written to by software at the same time it would be altered by a compare event, the software write will have priority. In this case the hardware-triggered change will not become effective.
Only capture/compare channels 16...19 and 24...27 are connected to pins.

Compare Mode 2

Compare mode 2 is an interrupt-only mode similar to compare mode 0, but only one interrupt request per timer period will be generated. Compare mode 2 is selected for register CCx by setting bit field CCMODx of the corresponding mode control register to ‘110_B’.

When a match is detected in compare mode 2 for the first time within a timer period, the interrupt request flag CCxIR is set to ‘1’. The corresponding port pin is not affected and can be used for general purpose IO. However, after the first match has been detected in this mode, all further compare events within the same timer period are disabled for compare register CCx until the allocated timer overflows. This means, that after the first match, even when the compare register is reloaded with a value higher than the current timer value, no compare event will occur until the next timer period.

In the example below, the compare value in register CCx is modified from cv1 to cv2 after compare event #1. Compare event #2, however, will not occur until the next period of timer Ty.

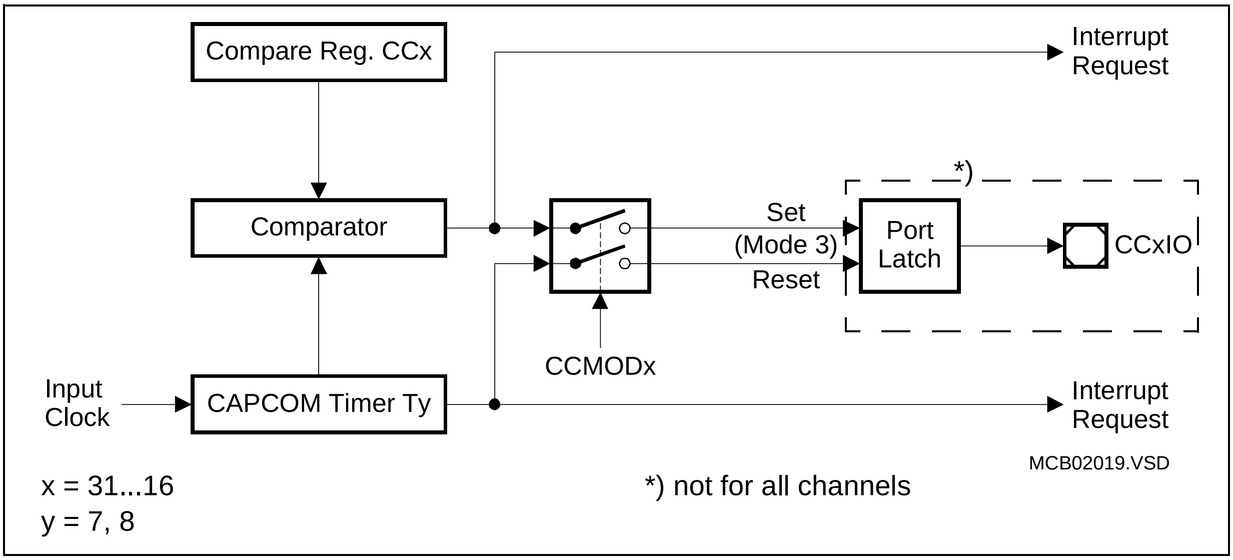


Figure 16-8 Compare Mode 2 and 3 Block Diagram

Note: The port latch and pin remain unaffected in compare mode 2.

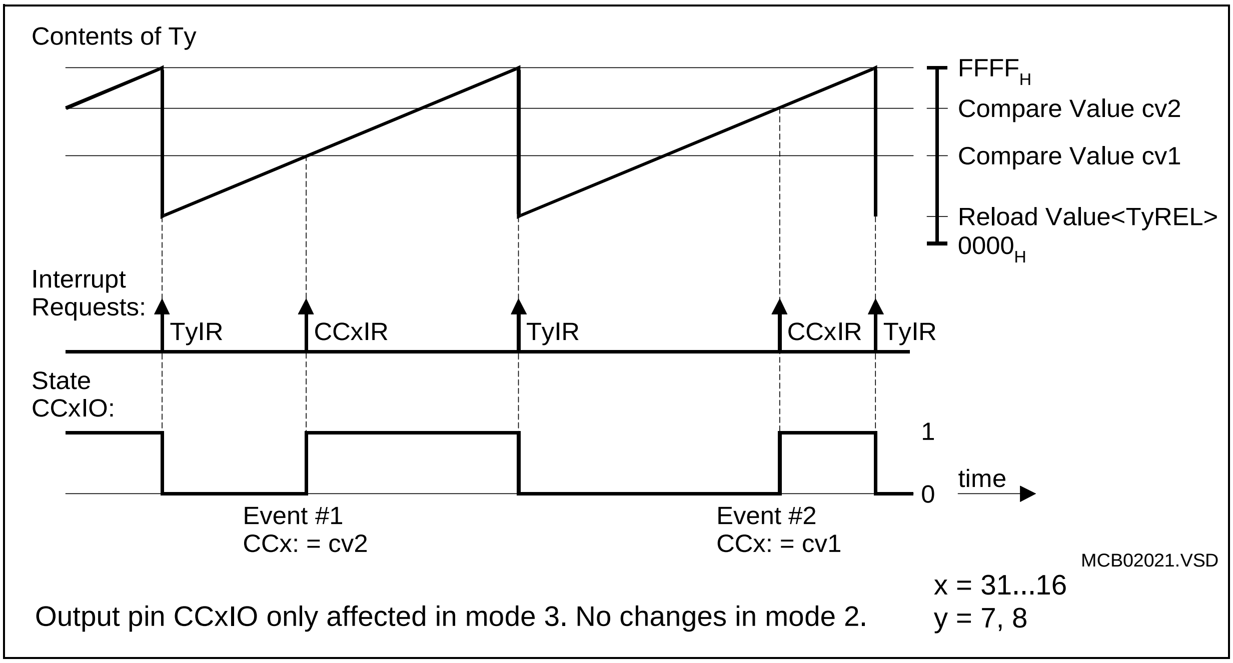


Figure 16-9 Timing Example for Compare Modes 2 and 3

Compare Mode 3

Compare mode 3 is selected for register CCx by setting bit field CCMODx of the corresponding mode control register to ‘111_B’. In compare mode 3 only one compare event will be generated per timer period.

When the first match within the timer period is detected the interrupt request flag CCxIR is set to ‘1’ and also the output pin CCxIO (alternate port function) will be set to ‘1’. The pin will be reset to ‘0’, when the allocated timer overflows.

If a match was found for register CCx in this mode, all further compare events during the current timer period are disabled for CCx until the corresponding timer overflows. If, after a match was detected, the compare register is reloaded with a new value, this value will not become effective until the next timer period.

In order to use the respective port pin as compare signal output pin CCxIO for compare register CCx in compare mode 3 this port pin must be configured as output, i.e. the corresponding direction control bit must be set to ‘1’. With this configuration, the initial state of the output signal can be programmed or its state can be modified at any time by writing to the port output latch.

In compare mode 3 the port latch is set upon a compare event and cleared upon a timer overflow (see Timing Example above).

However, when compare value and reload value for a channel are equal the respective interrupt requests will be generated, only the output signal is not changed (set and clear would coincide in this case).

Note: If the port output latch is written to by software at the same time it would be altered by a compare event, the software write will have priority. In this case the hardware-triggered change will not become effective.
Only capture/compare channels 16...19 and 24...27 are connected to pins.

Double-Register Compare Mode

In double-register compare mode two compare registers work together to control one output pin. This mode is selected by a special combination of modes for these two registers.

For double-register mode the 16 capture/compare registers of each CAPCOM unit are regarded as two banks of 8 registers each. Registers CC16...CC23 form bank 1 while registers CC24...CC31 form bank 2 (respectively). For double-register mode a bank 1 register and a bank 2 register form a register pair. Both registers of this register pair operate on the pin associated with the bank 1 register (pins CC16IO...CC19IO are available).

The relationship between the bank 1 and bank 2 register of a pair and the effected output pins for double-register compare mode is listed in the table below.

Table 16-4 Register Pairs for Double-Register Compare Mode

CAPCOM2 Unit		
Register Pair		**Associated Output Pin**
Bank 1	**Bank 2**	
CC16	CC24	CC16IO
CC17	CC25	CC17IO
CC18	CC26	CC18IO
CC19	CC27	CC19IO
CC23...CC20	CC31...CC28	-----

The double-register compare mode can be programmed individually for each register pair. In order to enable double-register mode the respective bank 1 register (see table) must be programmed to compare mode 1 and the corresponding bank 2 register (see table) must be programmed to compare mode 0.
If the respective bank 1 compare register is disabled or programmed for a mode other than mode 1 the corresponding bank 2 register will operate in compare mode 0 (interrupt-only mode).

In the following, a bank 2 register (programmed to compare mode 0) will be referred to as CCz while the corresponding bank 1 register (programmed to compare mode 1) will be referred to as CCx.

When a match is detected for one of the two registers in a register pair (CCx or CCz) the associated interrupt request flag (CCxIR or CCzIR) is set to ‘1’ and pin CCxIO corresponding to bank 1 register CCx is toggled. The generated interrupt always corresponds to the register that caused the match.

Note: If a match occurs simultaneously for both register CCx and register CCz of the register pair pin CCxIO will be toggled only once but two separate compare interrupt requests will be generated, one for vector CCxINT and one for vector CCzINT.

In order to use the respective port pin as compare signal output pin CCxIO for compare register CCx in double-register compare mode, this port pin must be configured as output, i.e. the corresponding direction control bit must be set to '1'. With this configuration, the output pin has the same characteristics as in compare mode 1.

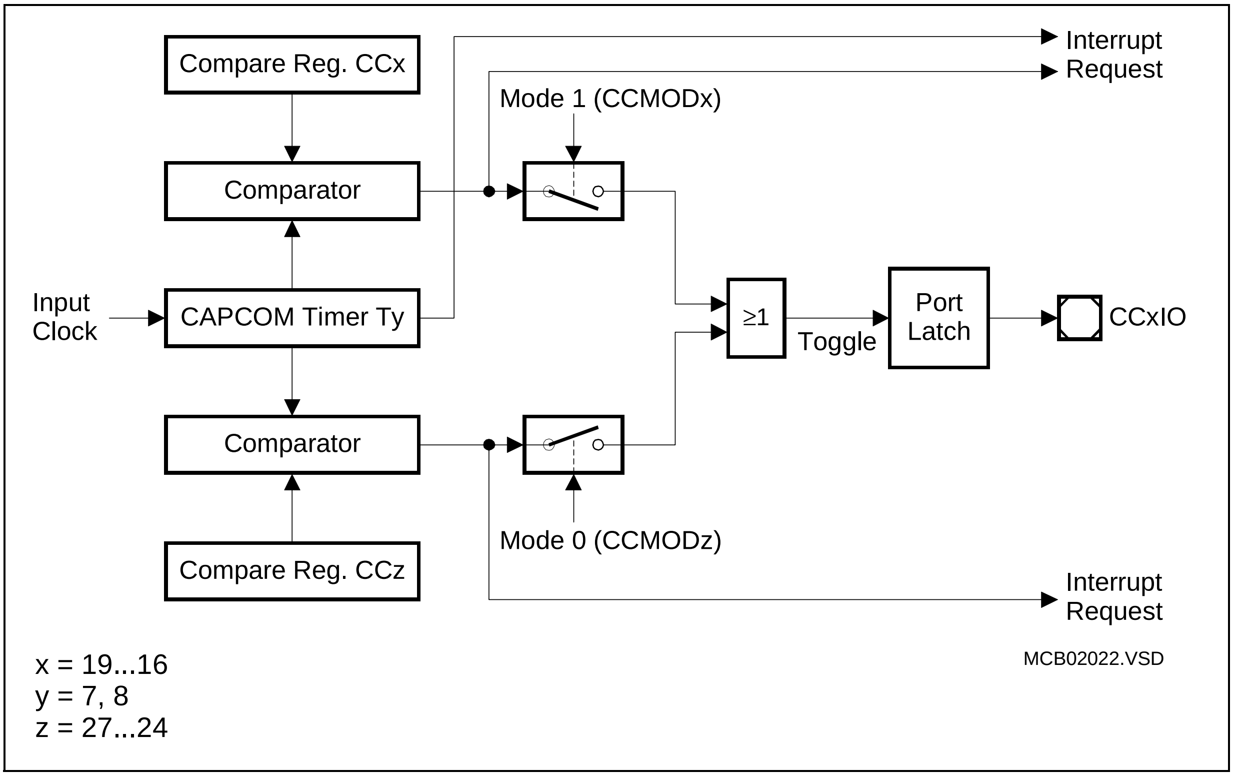


Figure 16-10 Double-Register Compare Mode Block Diagram

In this configuration example, the same timer allocation was chosen for both compare registers, but each register may also be individually allocated to one of the two timers of the respective CAPCOM unit. In the timing example for this compare mode (below) the compare values in registers CCx and CCz are not modified.

Note: The pins CCzIO (which do not serve for double-register compare mode) may be used for general purpose IO.

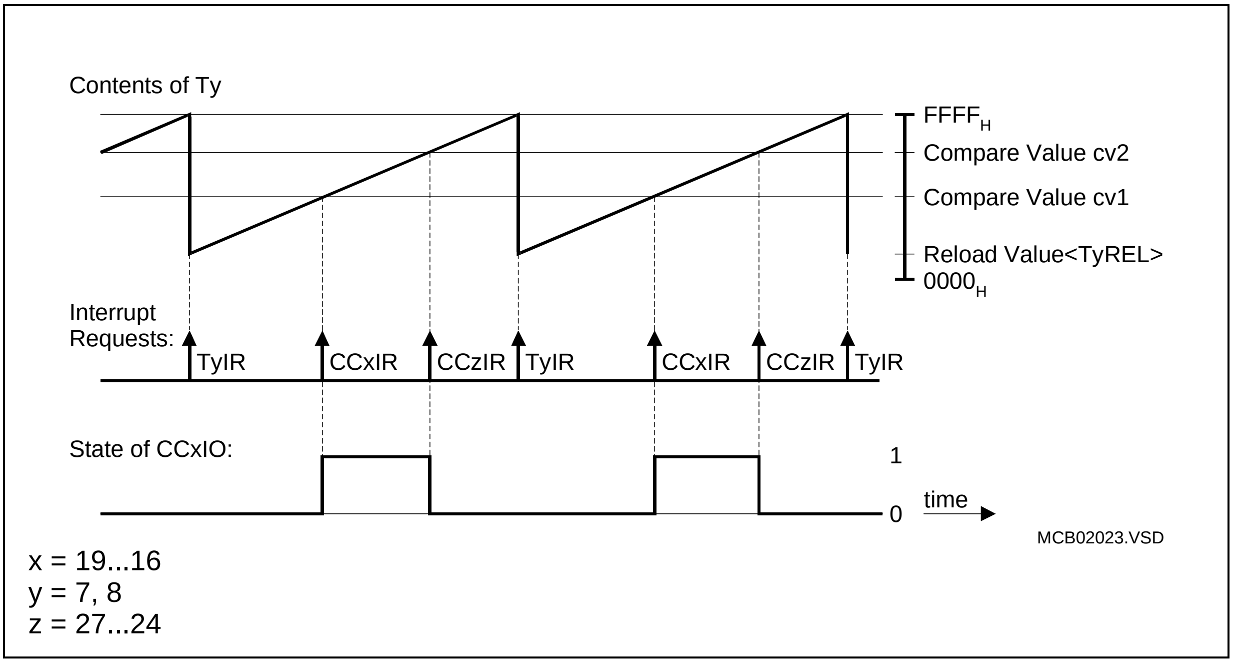


Figure 16-11 Timing Example for Double-Register Compare Mode

Note: Only on channel pairs with an associated output pin Double-Register Compare Mode is reasonable.

16.6 Capture/Compare Interrupts

Upon a capture or compare event, the interrupt request flag CCxIR for the respective capture/compare register CCx is set to ‘1’. This flag can be used to generate an interrupt or trigger a PEC service request when enabled by the interrupt enable bit CCxIE.

Capture interrupts can be regarded as external interrupt requests with the additional feature of recording the time at which the triggering event occurred (see also section “External Interrupts”).

Each of the 16 capture/compare registers has its own bitaddressable interrupt control register (CC31IC...CC16IC) and its own interrupt vector (CC31INT...CC16INT). These registers are organized the same way as all other interrupt control registers. The figure below shows the basic register layout, and the table lists the associated addresses.

CCxIC
CAPCOM Intr. Ctrl. Reg. **ESFR (*See Table*)** **Reset value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								CCx IR	**CCx IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

Table 16-5 CAPCOM Unit Interrupt Control Register Addresses

CAPCOM2 Unit		
Register Name	**Address**	**Register Space**
CC16IC	F160$_H$ / B0$_H$	ESFR
CC17IC	F162$_H$ / B1$_H$	ESFR
CC18IC	F164$_H$ / B2$_H$	ESFR
CC19IC	F166$_H$ / B3$_H$	ESFR
CC20IC	F168$_H$ / B4$_H$	ESFR
CC21IC	F16A$_H$ / B5$_H$	ESFR
CC22IC	F16C$_H$ / B6$_H$	ESFR
CC23IC	F16E$_H$ / B7$_H$	ESFR
CC24IC	F170$_H$ / B8$_H$	ESFR
CC25IC	F172$_H$ / B9$_H$	ESFR
CC26IC	F174$_H$ / BA$_H$	ESFR
CC27IC	F176$_H$ / BB$_H$	ESFR
CC28IC	F178$_H$ / BC$_H$	ESFR
CC29IC	F184$_H$ / C2$_H$	ESFR
CC30IC	F18C$_H$ / C6$_H$	ESFR
CC31IC	F194$_H$ / CA$_H$	ESFR

17 The Capture/Compare Unit CAPCOM6

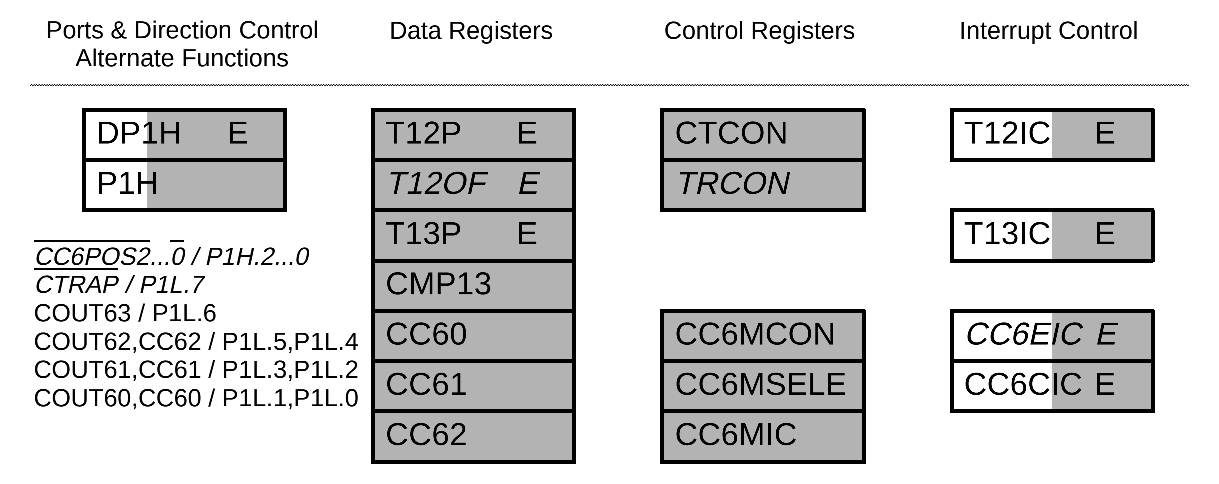
The CAPCOM6 unit of the C164 has been designed for applications which have a demand for digital signal generation and/or event capturing (e.g. pulse width modulation, pulse width measuring). It supports generation and control of timing sequences on up to three 16-bit capture/compare channels plus one 10-bit compare channel.

In compare mode the CAPCOM6 unit provides two output signals per 16-bit channel which may have inverted polarity and non-overlapping pulse transitions. The 10-bit compare channel can generate a single PWM output signal and is further used to modulate the capture/compare output signals. The compare timers T12 (16-bit) and T13 (10-bit) are free running timers which are clocked by the prescaled CPU clock.

For motor control applications both subunits may generate versatile multichannel PWM signals which are basically either controlled by compare timer T12 or by a typical hall sensor pattern at the interrupt inputs. This operating mode is called block commutation (available only in devices with a full function CAPCOM6).

In capture mode the contents of compare timer T12 is stored in the capture registers upon a programmable signal transition at pins CC6x.

From the programmer’s point of view, the term ’CAPCOM unit’ refers to a set of SFRs which are associated with this peripheral, including the port pins which may be used for alternate input/output functions including their direction control bits.



DP1H	Port P1H Direction Control Register	CC60...62	CAPCOM6 Register 0...2
P1H	Port P1H Data Register	*TRCON*	*CAPCOM6 Trap Control Register*
		CC6MCON	CAPCOM6 Mode Control Register
TxP	CAPCOM6 Timer x Period Register	CC6MSEL	CAPCOM6 Mode Select Register
T12OF	*CAPCOM6 Timer T12 Offset Register*	CC6MIC	CAPCOM6 Mode Intr. Control Register
CMP13	CAPCOM6 Timer T13 Compare Register	*CC6EIC*	*CAPCOM6 Emergency Intr. Control Reg.*
CTCON	CAPCOM6 Timer Control Register	CC6CIC	CAPCOM6 Channel Intr. Control Register

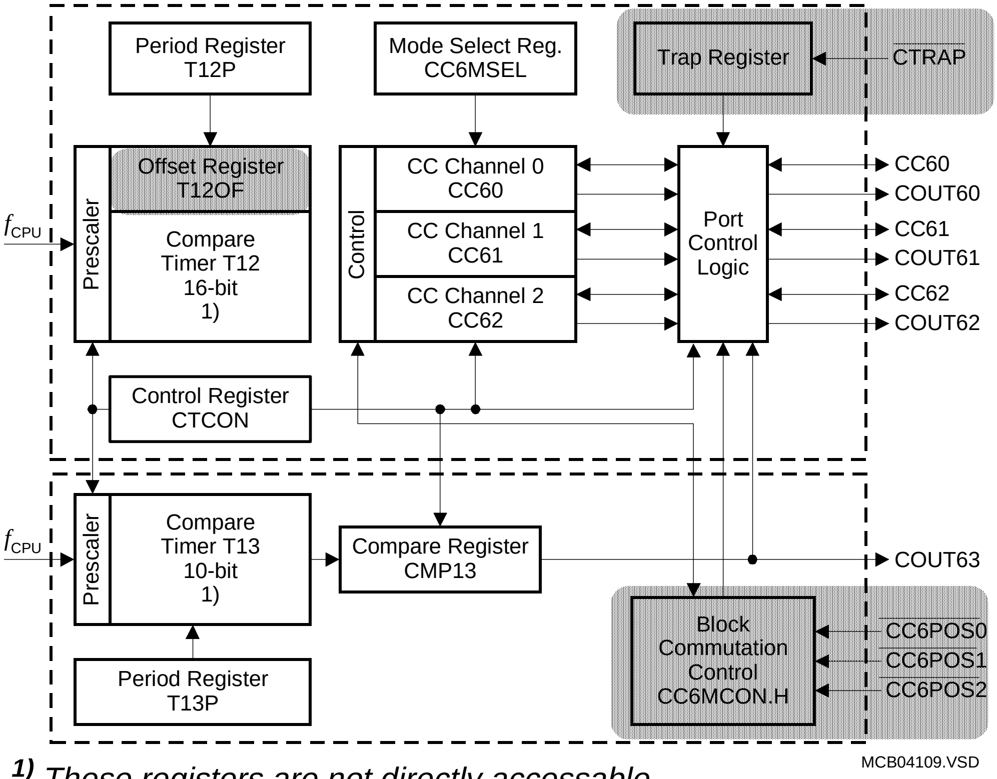
Note: The *resources marked in italic* are available only in the *full function CAPCOM6*.

Figure 17-1 SFRs and Port Pins associated with the CAPCOM6 Unit

The three 16-bit capture/compare channels are driven via timer T12 and can control two output lines each (see Port Control Logic). The offset register T12OF (full function module only) allows to shift the switching points of the COUT6x output line of each channel by shifting the respective compare value.

The 10-bit compare channel is driven via timer T13 and can control one output line.

Additional control logic allows the combination of the capture/compare channel outputs with the compare channel output or with external signals. Thus flexible and complex output patterns can be generated automatically, i.e. with very little or no CPU action at all.



Note: [1] These registers are not directly accessable. The period and offset registers are loading a value into the timer registers.

Figure 17-2 CAPCOM6 Block Diagram

Two basic operating modes are supported:

In **Edge Aligned Mode** the compare timer counts up starting at 0000_H. Upon reaching the period value stored in register TxP the timer is cleared and repeats counting up. At this time also the output signals are switched to their passive state. Edge aligned mode is supported by both compare timers, T12 and T13.

In **Center Aligned Mode** the compare timer T12 counts up starting at 0000_H. Upon reaching the period value stored in register T12P the count direction is reversed and the timer counts down. The output signals are switched to their active/passive state upon a match with the compare value while counting up/down. Center aligned mode is only supported by compare timer T12.

The compare timers T12 and T13 are free running timers which are clocked with a programmable frequency of f_{CPU} to $f_{CPU}/128$.

The respective output signals are changed (if appropriate) when the timer reaches the programmed compare value. For switching the output signals COUT60...COUT62 the timer contents plus the offset value are compared against the compare value.

Timer T12 can operate in edge aligned or in center aligned PWM mode (see figure below), with or without a constant edge delay (a or b in the figure).
Timer T13 can operate in edge aligned mode without edge delay.

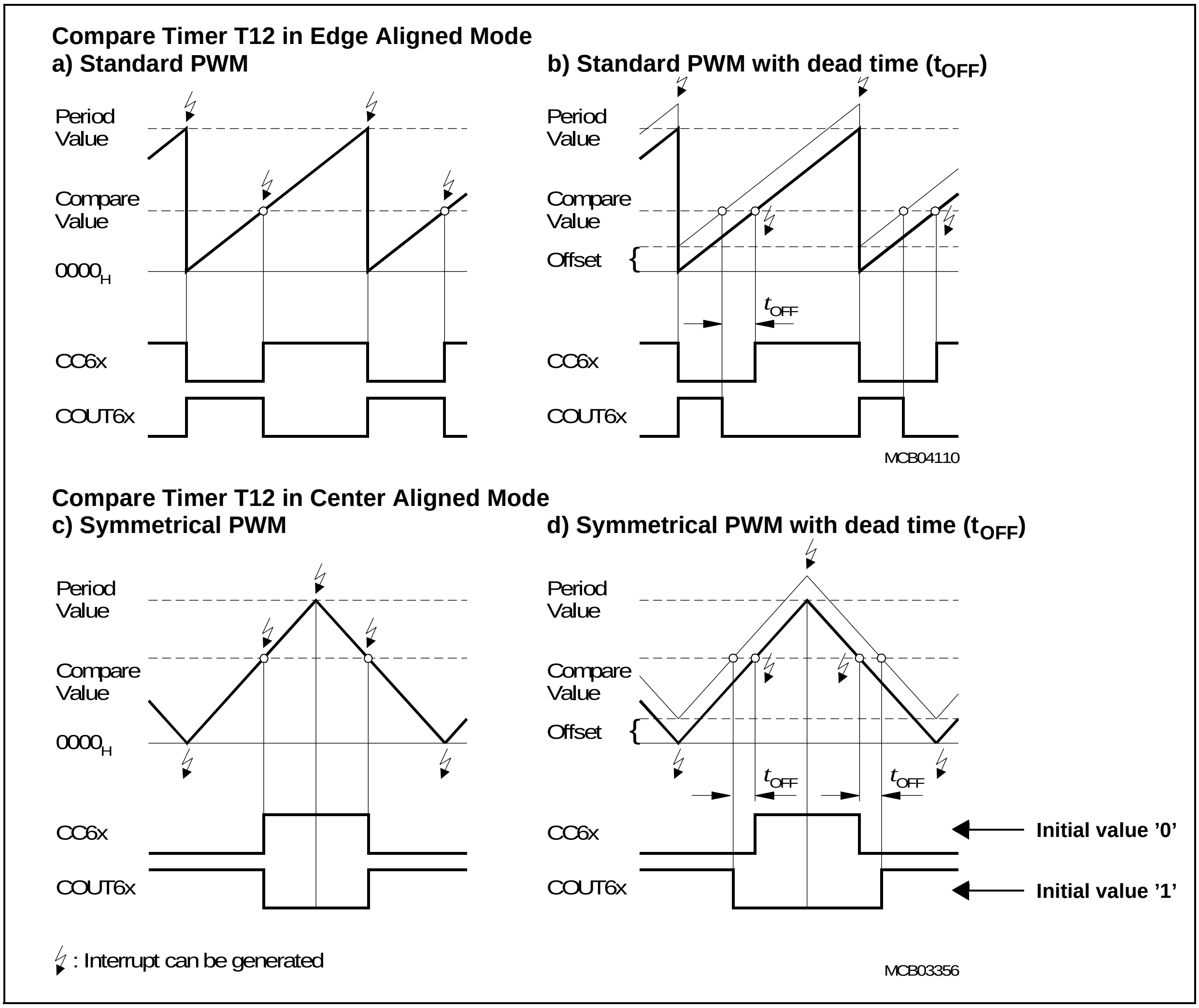
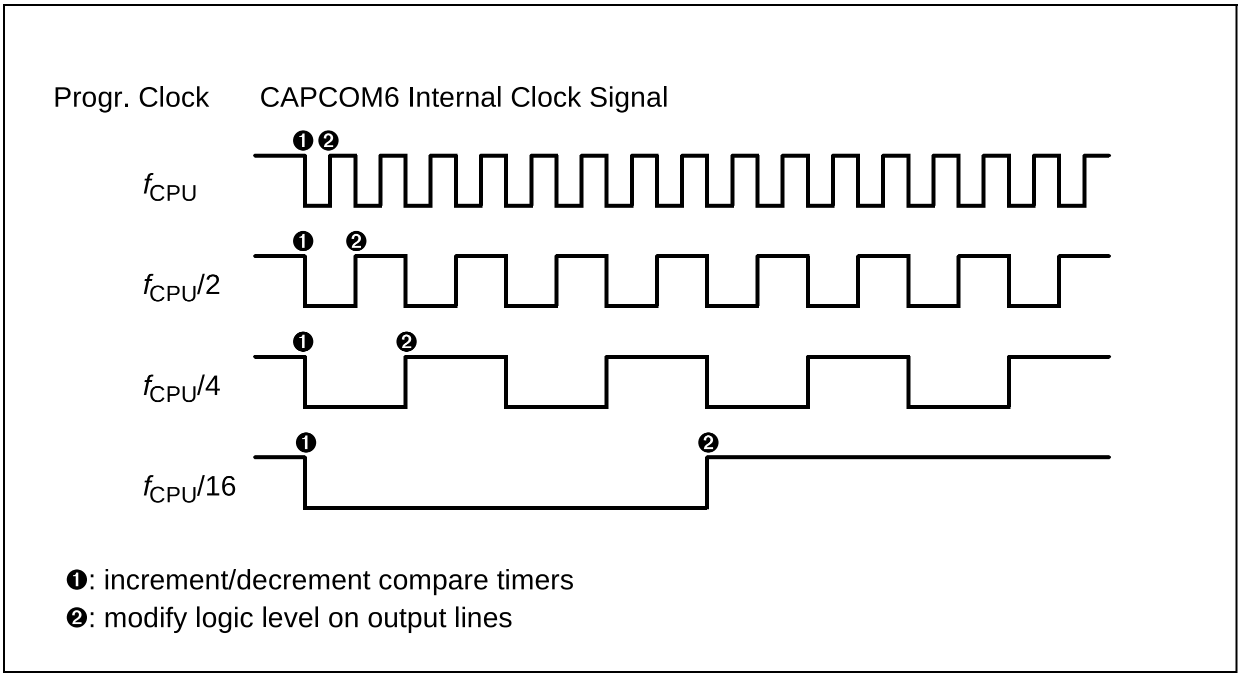

Compare Timer T12 in Edge Aligned Mode
a) Standard PWM
b) Standard PWM with dead time (t_{OFF})
Period Value
Compare Value
0000_H
Offset
t_{OFF}
CC6x
COUT6x
MCB04110
Compare Timer T12 in Center Aligned Mode
c) Symmetrical PWM
d) Symmetrical PWM with dead time (t_{OFF})
Initial value '0'
Initial value '1'
: Interrupt can be generated
MCB03356

Figure 17-3 CAPCOM6 Basic Operating Modes

17.1 Clocking Scheme

The CAPCOM6 unit operates on a programmable clock (f_{CPU}...f_{CPU}/128). This internal clock signal is used to control all actions within the unit.

The **falling edge** modifies the compare timers, the **rising edge** modifies the output signals (if required).



❶: increment/decrement compare timers
❷: modify logic level on output lines

Figure 17-4 CAPCOM6 Internal Clocking Scheme

17.2 Output Signal Level Control

The output signals generated by the CAPCOM6 unit are characterized by the duration of their active and passive phases, which define the signals' period and duty cycle. In order to adapt these output signals to the requirements of a specific application the logic level of the passive state for each signal can be selected via register CC6MCON.

When using the trap function, the outputs are switched to their trap level upon the activation of an external (emergency) signal. The trap level is defined via the respective port output latches.

Note: Changing the state levels during operation of CAPCOM6 will immediately affect the output signals. It is rather recommended to define the output levels during initialization before the output signals are assigned and the CAPCOM6 unit is started.

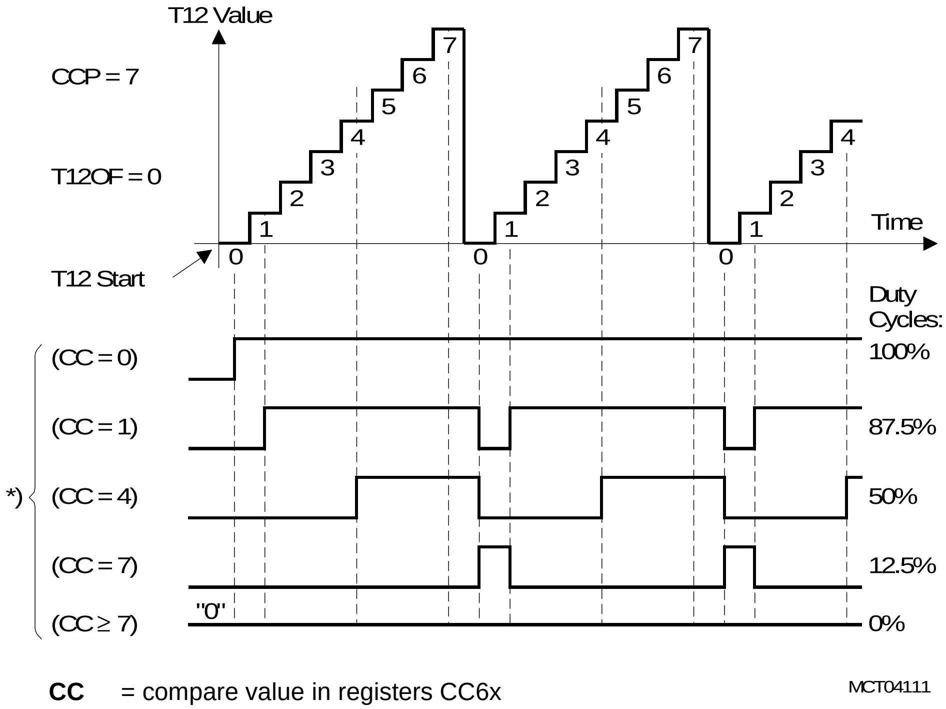
In burst and multi-channel modes the signals generated by the capture/compare channels may additionally be modulated by the signal generated by the 10-bit compare channel. This compare channel signal may optionally be inverted before modulating the other outputs. The compare channel's signal may be output on pin COUT63. This output function is enabled by bit ECT13O in register CTCON. If the output function is disabled COUT63 drives the defined passive level.

Note: Trap function and multi-channel modes are available in the full function module only.

17.3 Edge Aligned Mode

The compare timer counts up starting at 0000_H. When the timer contents match the respective compare value in register CC6x the associated output signal is switch to its active state. Upon reaching the period value stored in register TxP the timer is cleared and repeats counting up. At this time also the output signals are switched to their passive state.

In the figure below the selected edge offset is zero, therefore the output signal refers to CC6x and/or COUT6x.



CC = compare value in registers CC6x
CCP = period value in register T12P T13P
*) Output signal at pin CC6x or COUT6x selected as **active high**.
For an active low output the signals would appear inverted.

Figure 17-5 Operation in Edge Aligned Mode

The example above shows how to generate PWM output signals with duty cycles between 0% and 100%, including the corner values. The duty cycle directly corresponds to the programmed compare value. The indicated output signals can be output on the respective pin CC6x or COUT6x or both of them. The pin allocation is controlled via bitfields CMSELx in register CC6MSEL. Register CC6MCON selects the passive level for enabled outputs. The example above uses active high signals, i.e. the passive level is low (associated select bit is ’0’).

In the figure below a non-zero offset value is used. In this case the compare value is not compared with the timer contents directly but rather with timer contents plus offset. As a consequence the active edge of signal COUT6x is shifted against CC6x.

The figure shows some output signals that can be generated (compare value = ’3’):

a) Standard output signal, using T12 directly, active high.
b) Shifted output signal, using T12+T12OF, active high.
c) Same signal as b), but active low.
d) 0% output signal, compare value in CC6x > T12P+T12OF.
e) 100% output signal, compare value in CC6x = T12OF.

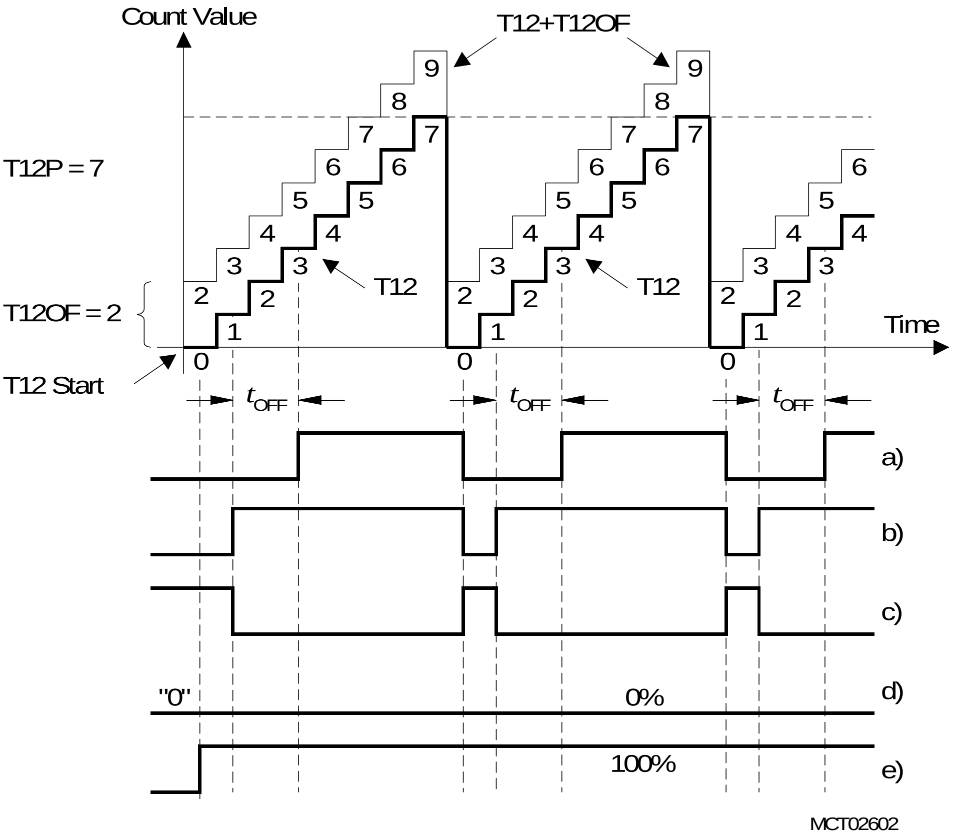


Figure 17-6 Operation with Non-zero Offset

Note: Offset operation is only available in the full function module.
It is possible only for the 3 capture/compare channels on timer T12. The compare channel on timer T13 does not provide an offset register and has no second output signal.

17.4 Center Aligned Mode

The 3 capture/compare channels associated with T12 may operate in center aligned mode. The compare timer T12 counts up starting at 0000_H. When the timer contents match the respective compare value in register CC6x the associated output signal CC6x is switched to its **active** state (while counting **up**). Upon reaching the period value stored in register T12P the count direction is reversed and the timer counts down. When the timer contents match the respective compare value in register CC6x the associated output signal CC6x is switched to its **passive** state (while counting **down**).

The output signals COUT6x are switched upon matches of register CC6x with T12+T12OF. Non-zero offset values shift the COUT6x edges symmetrically against the CC6x edges (see figure below). This allows the generation of non-overlapping signal pairs CC6x/COUT6x with arbitrary active levels. These signal pairs may e.g. be used to drive the high and low side switches of a power bridge without the risk of a branch shortcut (prevented by the programmable dead-time t_{OFF}, see figure below).

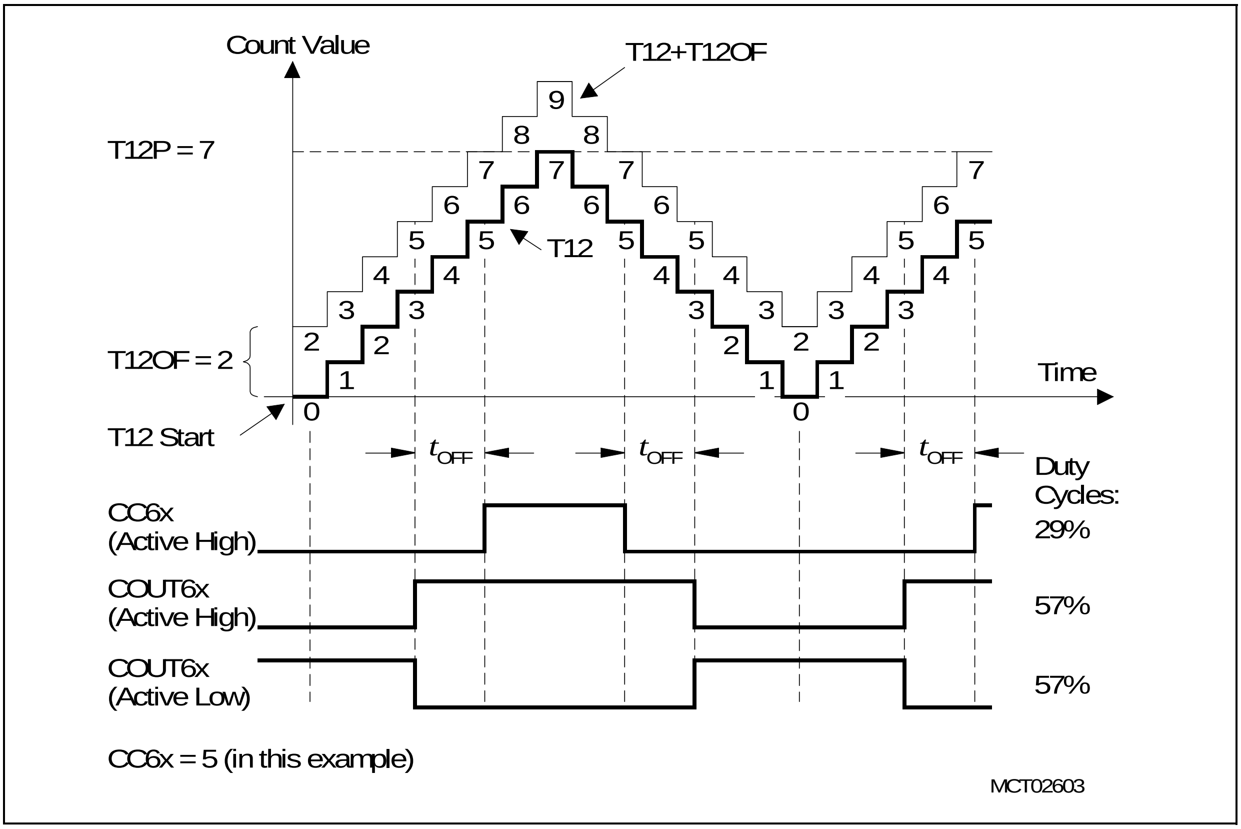


Figure 17-7 Operation in Center Aligned Mode

Note: In order to generate correct dead times for PWM signals the offset value stored in T12OF must be lower than the value stored in the compare registers.
The offset value affects all COUT6x outputs.
Dead time generation is available only in the full function module.

17.4.1 Timing Relationships

The resolution of the compare timers depends on the selected internal clock frequency. The period range of the output signals in turn depends on the actual timer resolution (minimum value) and on the timer and period values (maximum value). The table below lists the respective values for both compare timers for the possible clock selections.

Due to the internal operation the minimum possible output period is 2 internal clock cycles.

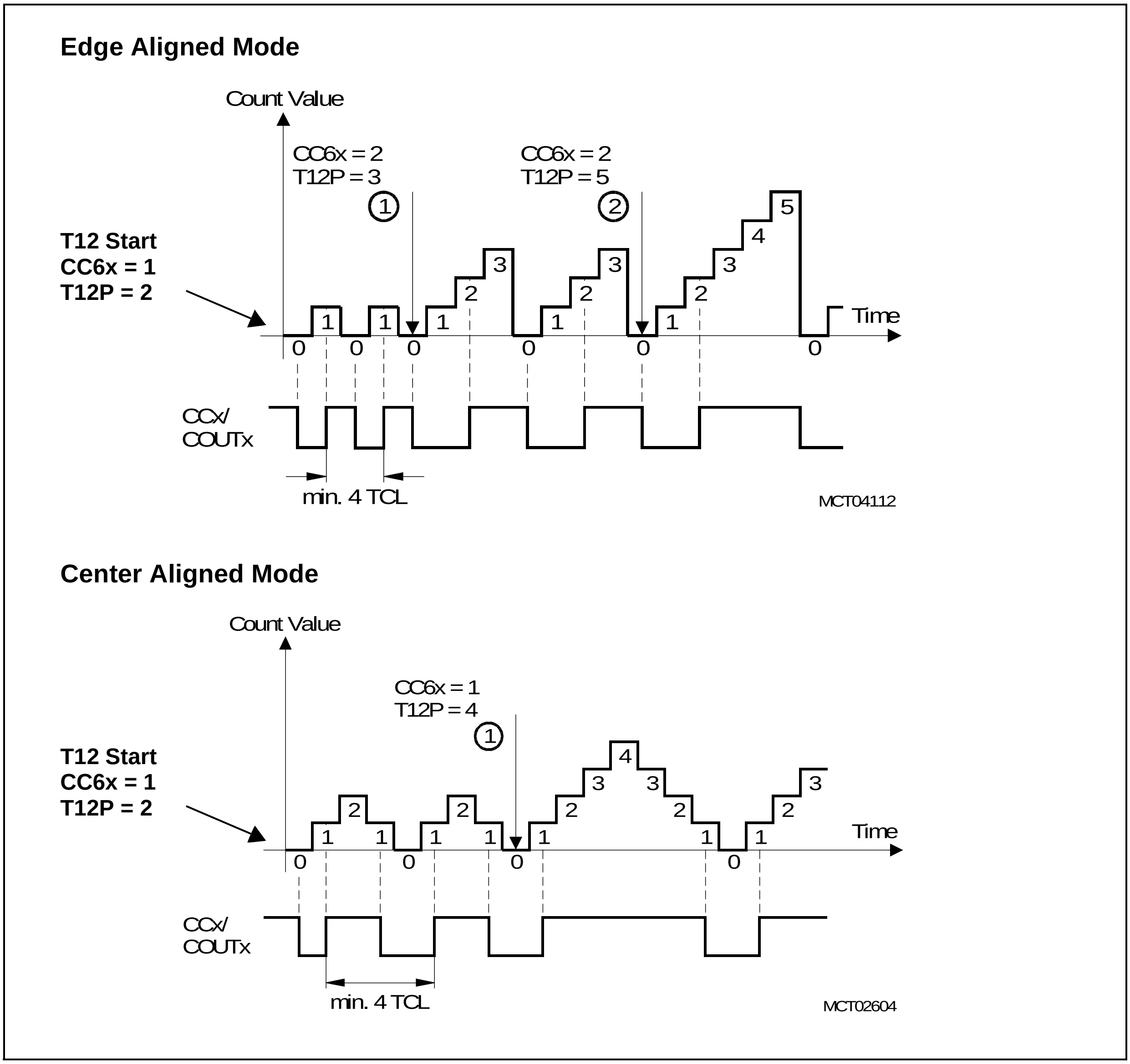


Figure 17-8 Operation in Center Aligned Mode

Table 17-1 Compare Timer Resolution and Period Range as Function of the Internal Clock @ f_{CPU}=20 MHz

Internal Clock	Cmp.Timer Resolution	Output Signal Period Range (Txmin. - T12max. / T13max.)	
		Edge Aligned Mode	Center Aligned Mode
f_{CPU}	50 ns	100ns - 3.28ms / 51.2µs	200ns - 6.55ms / 102.4µs
f_{CPU} / 2	100 ns	200ns - 6.55ms / 102.4µs	400ns - 13.11ms / 204.8µs
f_{CPU} / 4	200 ns	400ns - 13.11ms / 204.8µs	800ns - 26.21ms / 409.6µs
f_{CPU} / 8	400 ns	800ns - 26.21ms / 409.6µs	1.6µs - 52.43ms / 819.2µs
f_{CPU} / 16	800 ns	1.6µs - 52.43ms / 819.2µs	3.2µs - 104.86ms/ 1.64ms
f_{CPU} / 32	1.6 µs	3.2µs - 104.86ms/ 1.64ms	6.4µs - 209.72ms/ 3.28ms
f_{CPU} / 64	3.2 µs	6.4µs - 209.72ms/ 3.28ms	12.8µs - 419.43ms/ 6.55ms
f_{CPU} / 128	6.4 µs	12.8µs - 419.43ms/ 6.55ms	25.6µs - 838.86ms/ 13.1ms

Compare timer Tx period and duty cycle values can be calculated using the formulas below. In these formulas the following abbreviations are used :

pv = period value, stored in register TxP
ov = offset value, stored in register T12OF
cv = compare value, stored in register CC6x or CMP13

Note: For compare timer T13 only the output signal COUT63 in edge aligned mode is available.

Edge Aligned Mode:

$$\text{Period value} = pv + 1$$

$$\text{Duty cycle of CC6x outputs} = \left(1 - \frac{cv}{pv + 1}\right) * 100\ \%$$

$$\text{Duty cycle of COUT6x outputs} = \left(1 - \frac{cv - ov}{pv + 1}\right) * 100\ \%$$

Center Aligned Mode:

$$\text{Period value} = 2 * pv$$

$$\text{Duty cycle of CC6x outputs} = \left(1 - \frac{cv}{pv}\right) * 100\ \%$$

$$\text{Duty cycle of COUT6x outputs} = \left(1 - \frac{cv - ov}{pv}\right) * 100\ \%$$

17.5 Burst Mode

In burst mode the output signal COUT63 of the 10-bit compare channel modulates the active phases of the output signals COUT6x of the 3 capture/compare channels. Burst mode is not possible on the CC6x outputs. The modulating signal typically has a higher frequency than the modulated output channels. The figure below shows an example for a waveform generated in burst mode.

Burst mode is enabled for each capture/compare output separately by setting the respective bit CMSELx3 in register CC6MSEL.

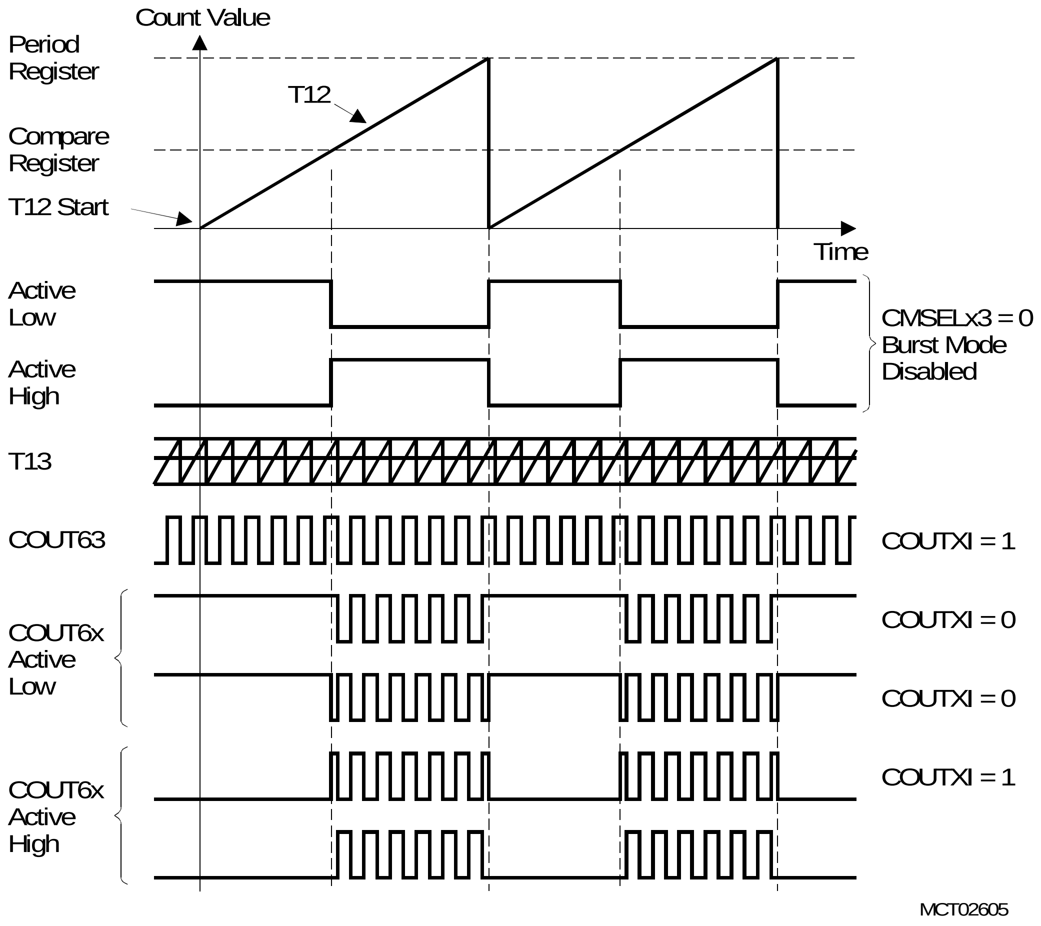


Figure 17-9 Operation in Burst Mode

17.6 Capture Mode

Each of the 3 capture/compare channels can individually be programmed for capture mode via bitfields CMSELx in register CC6MSEL. In capture mode the contents of timer T12 are copied to the channel's compare register CC6x upon a selectable transition (rising, falling or both) at the associated pin CC6x. Capture mode can be enabled in edge aligned mode as well as in center aligned mode. Interrupts may be generated selectively at each transition of the capture input signal.

Pins CC6x (used as inputs in capture mode) are sampled every CPU clock period.

When evaluating a series of capture events it must be respected that every capture event overwrites the previous value in the respective register CC6x. The control software must be designed to retrieve the capture values early enough.

17.7 Combined Multi-Channel Modes

Note: Multi-channel modes are available in the full function module only.

When operating in a combined multi-channel mode the output signals CC6x and COUT6x are controlled not only by the compare timers, but combined with additional conditions. Multi-channel modes are selected via register CC6MCON. In these modes a predefined signal pattern sequence is driven to the output lines.

Note: Compare timer T12 must be enabled (CT12R = '1') in order to enable proper operation of the multi-channel modes.

Multi-phase modes allow the effective generation of output signal patterns e.g. for 4...6 phase unipolar drives. The phase sequence can either be controlled automatically by T12 overflows or under software control.

Block Commutation mode is a special multi-channel mode which especially supports the control of brushless DC drives. In this mode the phase sequence is controlled by 3 input signals ($\overline{\text{CC6POSx}}$) which are generated by the drive (e.g. via hall sensors).

In all modes the output signals can be modulated during their active phases.

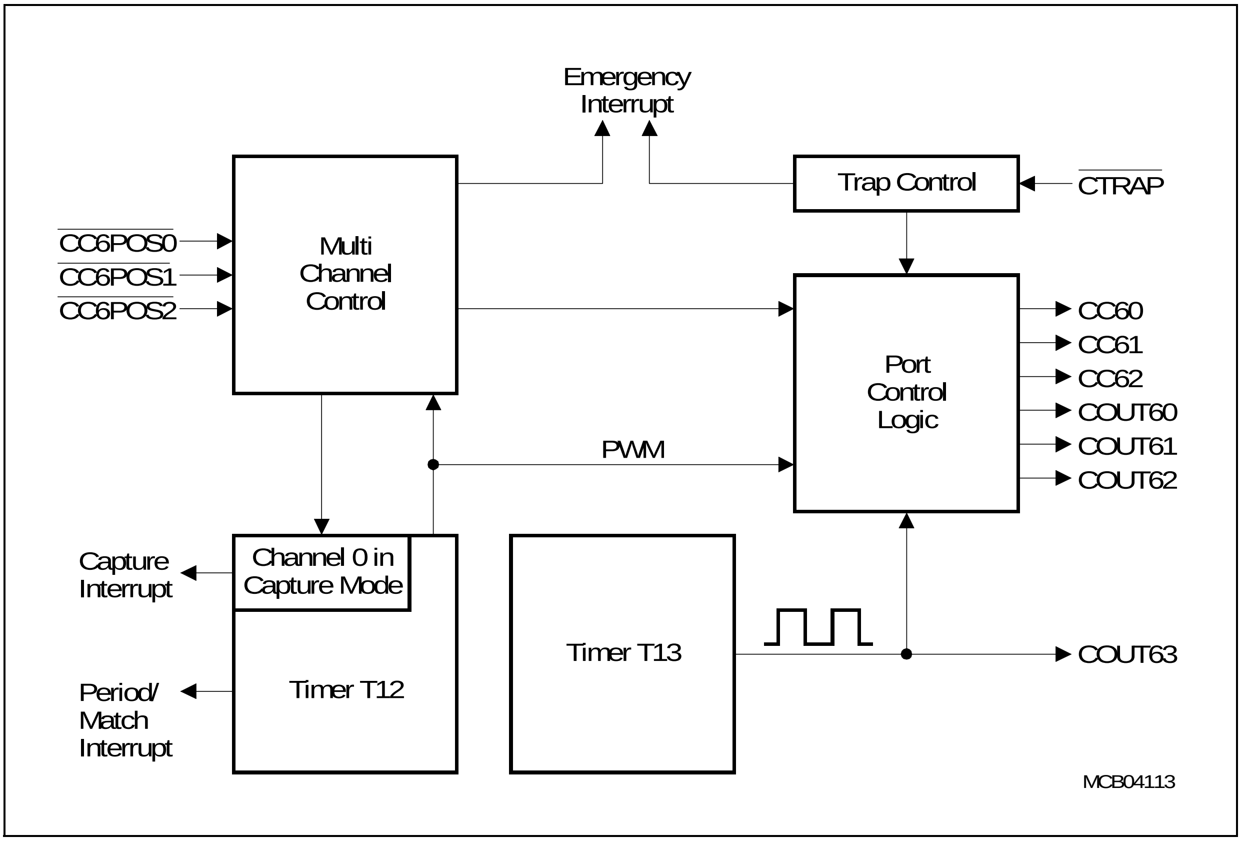
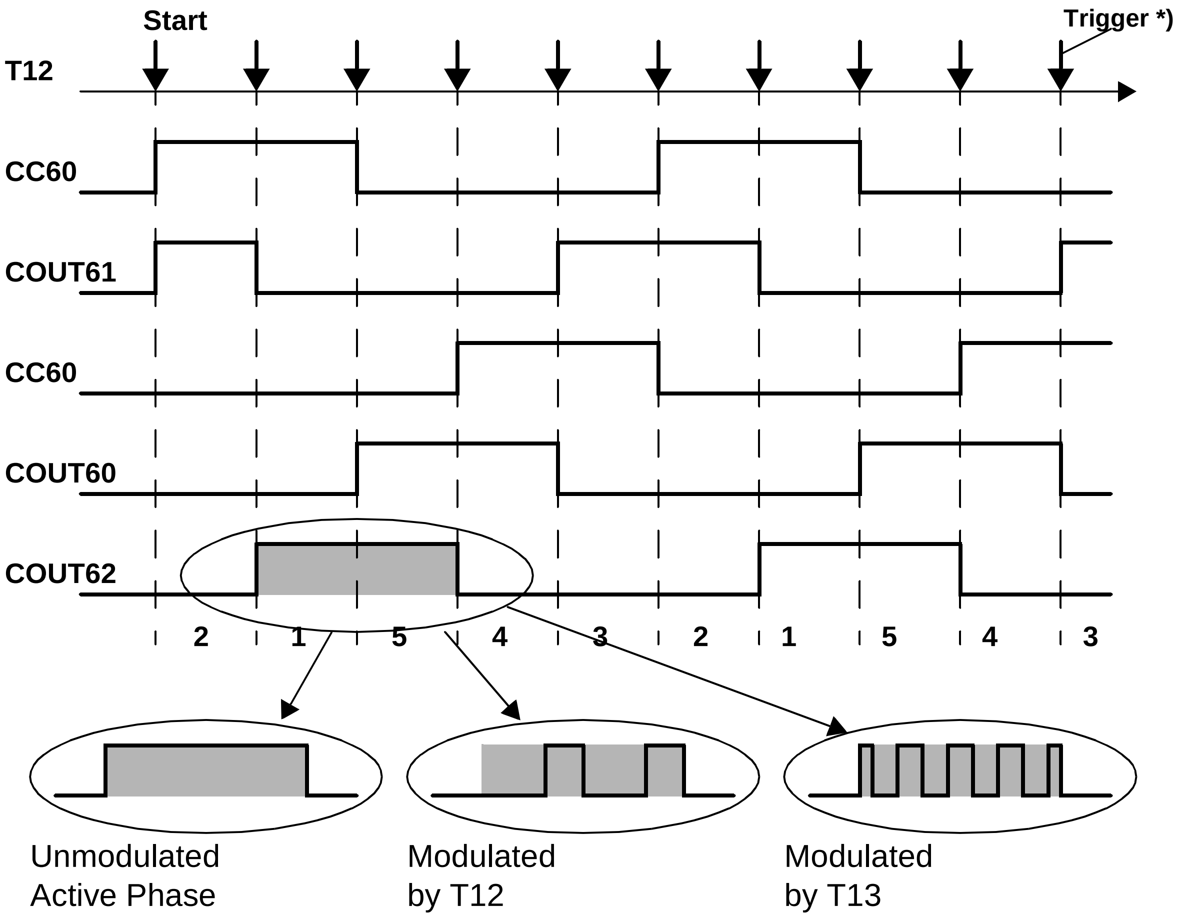


Figure 17-10 Multi-Channel Mode Control

17.7.1 Output Signals in Multi-Channel Mode

In multi-channel mode the output signals are mainly controlled by the selected phase sequence (see sequence tables below). Each output is active for two phases and remains passive for all other phases of a sequence.

The active phases of each output signal may additionally be modulated by T12 or T13. For unmodulated active phases timer T12 must operate with 100% duty cycle, i.e. its offset and compare registers must be cleared, and T13 modulation must be off, i.e. bits CMSELx3 must be cleared. T12 modulation is effective when T12's duty cycle is programmed below 100%, T13 modulation is enabled via bits CMSELx3 (see examples in the figure below).



*) The trigger that switches to the next phase may be a T12 overflow or a '1' being written to bit NMCS via software. In block commutation mode the trigger is represented by a change in the input pattern on pins $\overline{\text{CC6POSx}}$.

The shown waveforms are active high.

Figure 17-11 Basic 5-phase Multi-Channel Timing

Figure 17-11 on page 14 shows the 5-phase output waveforms as an example. For the other modes each passive phase is shortened or lengthened by one sequence phase, respectively.

The compare output signals are enabled according to the intended multi-phase mode.

The table below lists the required coding:

Table 17-2 Programming of Multi-Channel PWM Outputs

Multi-Channel PWM Mode	CMSEL2	CMSEL1	CMSEL0
Block commutation mode	011_B	011_B	011_B
4-phase multi-channel PWM	011_B	010_B	001_B
5-phase multi-channel PWM	011_B	010_B	011_B
6-phase multi-channel PWM	011_B	011_B	011_B

Note: Bit CMSELx3 (burst mode bit) defines if the signal at the COUT6x pins is modulated by compare timer T13 (CMSELx3='1') or not. T13 modulation may be combined with T12 modulation.

Phase Sequence Tables

The following tables list the phase sequences for the different multi-phase modes. The sequence is defined via the follower state for each state and also the output levels for each state are listed.

The states of a phase sequence are switched...

- **Automatically** upon a T12 overflow
- **Software controlled** by setting bit NMCS in register CC6MSEL.
  Bit ESMC = '1' enables the software controlled state switching and disables switching on T12 overflows.

Note: The actual logic levels for active and passive state are defined in register CC6MCON.
In 4-phase, 5-phase and 6-phase multi-channel PWM mode all output signals can be modulated by timer T12 or timer T13 during their active phases.

Table 17-3 4-phase PWM Sequence Table

State	Output Level Definition (for actual state)						Follower State (for BCM=...)			
	CC60	COUT61	CC62	COUT60	CC61	COUT62	01	10	00	11
0	passive	passive	passive	---	---	passive	2	1	0	5
1	ACTIVE	passive	passive	---	---	ACTIVE	4	2	0	5
2	ACTIVE	ACTIVE	passive	---	---	passive	1	3	0	5
3	passive	ACTIVE	ACTIVE	---	---	passive	2	4	0	5
4	passive	passive	ACTIVE	---	---	ACTIVE	3	1	0	5
5	passive	ACTIVE	passive	---	---	ACTIVE	2	1	0	5

Table 17-4 5-phase PWM Sequence Table

State	Output Level Definition (for actual state)						Follower State (for BCM=...)			
	CC60	COUT61	CC62	COUT60	CC61	COUT62	01	10	00	11
0	passive	passive	passive	passive	---	passive	2	1	0	6
1	ACTIVE	passive	passive	passive	---	ACTIVE	5	2	0	6
2	ACTIVE	ACTIVE	passive	passive	---	passive	1	3	0	6
3	passive	ACTIVE	ACTIVE	passive	---	passive	2	4	0	6
4	passive	passive	ACTIVE	ACTIVE	---	passive	3	5	0	6
5	passive	passive	passive	ACTIVE	---	ACTIVE	4	1	0	6
6	passive	ACTIVE	passive	ACTIVE	---	ACTIVE	2	1	0	6

Table 17-5 6-phase PWM Sequence Table

State	Output Level Definition (for actual state)						Follower State (for BCM=...)			
	CC60	COUT61	CC62	COUT60	CC61	COUT62	01	10	00	11
0	passive	passive	passive	passive	passive	passive	2	1	0	7
1	ACTIVE	ACTIVE	passive	passive	passive	passive	6	2	0	7
2	passive	ACTIVE	ACTIVE	passive	passive	passive	1	3	0	7
3	passive	passive	ACTIVE	ACTIVE	passive	passive	2	4	0	7
4	passive	passive	passive	ACTIVE	ACTIVE	passive	3	5	0	7
5	passive	passive	passive	passive	ACTIVE	ACTIVE	4	6	0	7
6	ACTIVE	passive	passive	passive	passive	ACTIVE	5	1	0	7
7	passive	ACTIVE	passive	ACTIVE	passive	ACTIVE	2	1	0	7

17.7.2 Block Commutation Mode

Block commutation mode is a special variation of the multi-channel modes where the phase sequence is not controlled internally but rather by the 3 input signals CC6POS2...0. The state of the 6 output signals is derived from the pattern present on the input signals. The table below summarizes the possible combinations.

Table 17-6 Block Commutation Sequence Table

Block Comm. Mode (BCM)	Control Inputs CC6POS...			Output Level Definition (for actual state)					
	0	**1**	**2**	**CC60**	**CC61**	**CC62**	**COUT60**	**COUT61**	**COUT62**
Rotate Left	1	0	1	passive	passive	ACTIVE	ACTIVE	passive	passive
	1	0	0	passive	ACTIVE	passive	ACTIVE	passive	passive
	1	1	0	passive	ACTIVE	passive	passive	passive	ACTIVE
	0	1	0	ACTIVE	passive	passive	passive	passive	ACTIVE
	0	1	1	ACTIVE	passive	passive	passive	ACTIVE	passive
	0	0	1	passive	passive	ACTIVE	passive	ACTIVE	passive
Rotate Right	1	1	0	ACTIVE	passive	passive	passive	ACTIVE	passive
	1	0	0	ACTIVE	passive	passive	passive	passive	ACTIVE
	1	0	1	passive	ACTIVE	passive	passive	passive	ACTIVE
	0	0	1	passive	ACTIVE	passive	ACTIVE	passive	passive
	0	1	1	passive	passive	ACTIVE	ACTIVE	passive	passive
	0	1	0	passive	passive	ACTIVE	passive	ACTIVE	passive
Rotate Left [1]	0	0	0	passive	passive	passive	passive	passive	passive
Rotate Right	1	1	1	passive	passive	passive	passive	passive	passive
Slow Down	X	X	X	passive	passive	passive	ACTIVE	ACTIVE	ACTIVE
Idle [2]	X	X	X	passive	passive	passive	passive	passive	passive

[1] If one of these two input signal combinations is detected in rotate left or rotate right mode, bit BCERR is set. If enabled an emergency interrupt is generated. When these (error) states are encountered, the idle state is entered immediately.

[2] Idle state is entered when a “wrong follower“ is detected (if bit BCEM=’1’), or in case of an illegal input pattern (see note 1). When idle state is entered the BCERR flag is always set.
Idle state can only be left when the BCERR flag is cleared by software.

In block commutation mode CAPCOM channel 0 is automatically configured for capture mode. Any signal transition at inputs $\overline{\text{CC6POS2...0}}$ generates a capture pulse for CAPCOM channel 0 and sets the interrupt request flag CC0R. A rising edge at output pin CC60 does not generate an interrupt request in block commutation mode.

The values provide a measure for the rotation speed of the connected drive. When evaluating the values captured from the free-running timer T12, the timer must not be stopped, as this disturbs the operation of block commutation mode.

Note: Modulation of the active phase via T12 is not supported. PWM via T13 is possible on COUT6x.
The block commutation input signals are available for the full function module only.

17.8 Trap Function

The trap function provides very efficient means to protect external circuitry which is connected to the CAPCOM6's output lines. The trap function is controlled by register TRCON and triggered by the input signal $\overline{\text{CTRAP}}$. The trigger function of input $\overline{\text{CTRAP}}$ can be enabled/disabled generally and the trap function can be applied to each capture/ compare channel (CC6x and COUT6x) individually. The figure below shows examples for a trap state in edge aligned mode and in center aligned mode.

The trap state is entered when $\overline{\text{CTRAP}}$ becomes active. The output signals are switched to their respective trap level (defined by port latch P1L) immediately, i.e. without any CPU activity. The trap flag (TRF in register TRCON) is set in order to signal this event to the software.
If bit CT12RES in register CTCON is set timer T12 is cleared upon a trap event, otherwise it continues counting. No more transitions on the output signals are generated any more, however.

The trap state is exited when T12 reaches the value 0000_H after input $\overline{\text{CTRAP}}$ has been sampled inactive. This „delay“ automatically resumes the generation of the programmed output signals after a trap event in a synchronized way.

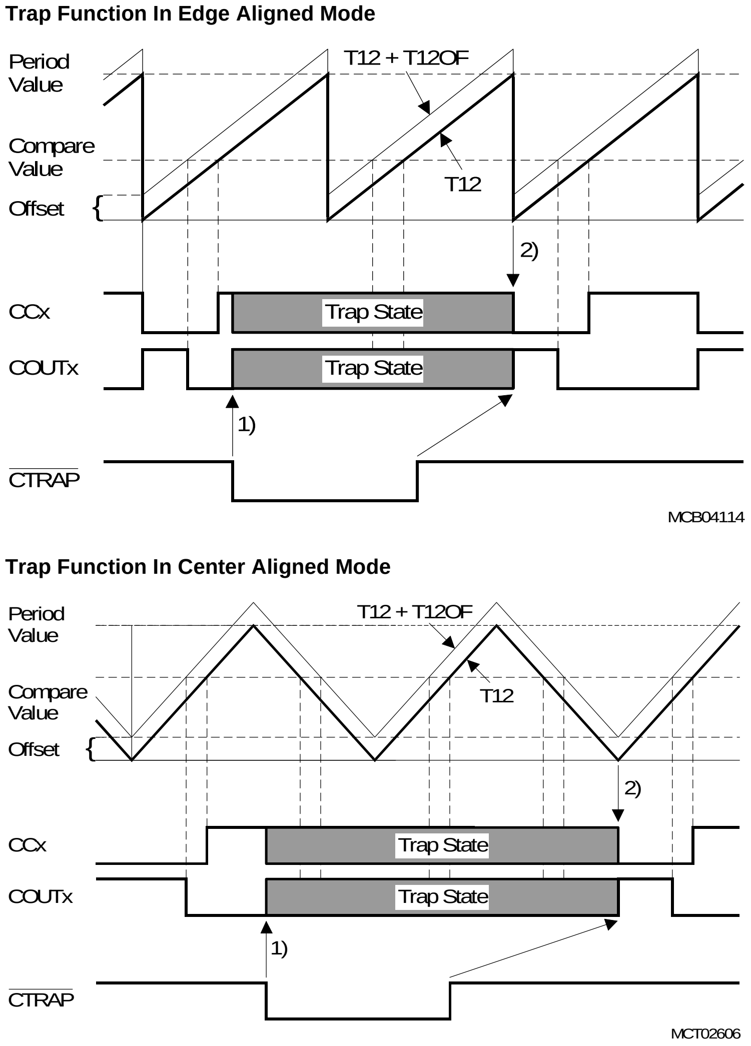
*Note: In block commutation mode trap state is exited when timer **T13** reaches 000_H (not T12).*

Notes

Reference Point 1):
Input CTRAP is activated, the outputs **immediately** switch to their **trap** level.

Reference Point 2):
Input CTRAP is inactive, the outputs switch to their programmed level **synchronized** to a new timer period.

Secure Trap State
During trap state the outputs CC6x and COUT6x which are selected for T13 modulation are not modulated with T13's output signal but rather with its initial value.



Trap Function In Edge Aligned Mode

Trap Function In Center Aligned Mode

Figure 17-12 Trap Function

Note: The TRAP function and the trap trigger input signal are available for the full function module only.

17.9 Register Description

The CAPCOM6 register set provides a number of control, data and status bits to control the operation of the two compare timers, the generation of the up to 7 output signals and the combination of submodules for multi-channel operation.

Note: The register bits which are available in the full function module only (not in the reduced version) are marked. This provides an immediate overview of the available registers and control/status bits in a specific derivative.

The table below summarizes the available registers. In the following the **control registers** are described in detail. Data registers (e.g. period or compare registers) are excluded from the detailled description. Please note that the timer registers (T12, T13) are not directly accessible.

Table 17-7 CAPCOM6 Register Summary

Name		Description	Address	Read
T12P	**E**	Timer T12 period register	F030$_H$ / 18$_H$	Sh.L.
T12OF	**E**	Timer T12 offset register	F034$_H$ / 1A$_H$	Sh.L.
T13P	**E**	Timer T13 period register	F032$_H$ / 19$_H$	Sh.L.
CMP13		Compare register for compare channel	FE36$_H$ / 1B$_H$	Sh.L.
CC60		Compare register for capture/compare channel 0	FE30$_H$ / 18$_H$	Reg.
CC61		Compare register for capture/compare channel 1	FE32$_H$ / 19$_H$	Reg.
CC62		Compare register for capture/compare channel 2	FE34$_H$ / 1A$_H$	Reg.
CTCON		Compare timer control register	FF30$_H$ / 98$_H$	Reg.
TRCON		Trap enable register	FF34$_H$ / 9A$_H$	Reg.
CC6MCON		CAPCOM6 mode control register	FF32$_H$ / 99$_H$	Reg.
CC6MSEL	**E**	CAPCOM6 mode select register	F036$_H$ / 1B$_H$	Reg.
CC6MIC		CAPCOM6 interrupt control register	FF36$_H$ / 9B$_H$	Reg.

Note: When reading these registers either the register itself or its shadow latch (see description below) is accessed. This is indicated in column „Read“.

In addition there are 4 interrupt node control registers associated with the CAPCOM6 unit, which are not part of the module, however.

Shadow Latches for Synchronous Update

The timer period, offset and compare values are written to shadow latches rather than to the actual registers. Also the initial value bits CCxI/COUTxI in register CC6MCON are equipped with shadow latches. Thus the values for a new output signal can be programmed without disturbing the currently generated signal(s). The transfer from the latches to the registers is enabled by setting the respective shadow latch transfer enable bit STEx in register CTCON.

If the transfer is enabled the shadow latches are copied to the respective registers as soon as the associated timer reaches the value zero the next time (being cleared in edge aligned mode or counting down from 1 in center aligned mode).

When timer T12 is operating in center aligned mode it will also copy the latches (if enabled) if it reaches the currently programmed period value (counting up).

After the transfer the respective bit STEx is cleared automatically.

Note: While T12/T13 is running the shadow latch transfer is controlled by bit STE12/13. While T12/T13 is stopped the shadow latch transfer is done automatically if bit CTRES12/13 is set, otherwise that latch values are not transferred.

Note: If a new compare value is written to the shadow latches while T12 is counting up, the new value must be smaller than the current period value. Otherwise no more matches will be detected and the output signals will not change any more.
If a compare value is written, while T12 is counting down, any value may be used.

CTCON

Compare Timer Control Reg. SFR ($FF30_H$/98_H) Reset value: 1010_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CT13P	**ECT13O**	**STE13**	**CT13RES**	**CT13R**	**CT13CLK**			**CTM**	**ETRP**	**STE12**	**CT12RES**	**CT12R**	**CT12CLK**		
rwh	rw	rwh	rw	rw	rw			rw	rw	rwh	rw	rw	rw		

Bit	Function
CTnCLK	**Compare Timer Tn Input Clock Select** Selects the input clock for timer T12 or T13 which is derived from the CPU clock: $f_{Tx} = f_{CPU} / 2^{<CTnCLK>}$. 000: $f_{Tx} = f_{CPU}$ ... 111: $f_{Tx} = f_{CPU} / 128$
CTnR	**Compare Timer Tn Run Bit** CTnR starts and stops timer Tn (T12 or T13). Together with bit CTnRES it controls Tn’s operation. 0: Timer Tn stops counting. If bit CTnRES = ’1’ timer Tn is cleared and the compare outputs are set to their defined idle state. 1: Timer Tn starts counting from its current value.
CTnRES	**Compare Timer Tn Reset Control** 0: No effect on timer Tn when it is stopped. 1: Timer Tn is cleared when it is stopped and the compare outputs are set to their defined idle state. *Note: for capture mode (T12 only):* *Clearing CT12R after a capture event while CT12RES = ’1’ will destroy the value stored in the capture register CC6x (all shadow registers are transparent). Leave CT12RES = ’0’ in capture mode.*

Bit	Function
STE12	**Timer T12 Shadow Latch Transfer Enable** 0: Transfer from the shadow latches to the initial value bits, and the period, compare and offset registers (T12P, CC6x, T12OF) of timer T12 is disabled. 1: Timer T12's initial value bits, and the period, compare and offset registers are loaded from their shadow latches when T12 reaches 0000_H (cleared in edge aligned mode, counting down in center aligned mode). In center aligned mode the registers are also loaded when T12 reaches the period value. *Note: STE12 is cleared by hardware after the shadow latch transfer.*
ETRP	**Emergency Trap Interrupt Enable** 0: The emergency interrupt for the CAPCOM6 trap signal is disabled. 1: The emergency interrupt for the CAPCOM6 trap signal is enabled.
CTM	**T12 Operating Mode** 0: Edge Aligned Mode: count up. 1: Center Aligned Mode: count up/down.
STE13	**Timer T13 Shadow Latch Transfer Enable** 0: Transfer from the shadow latches to the period and compare registers (CC62, CMPx) of timer T13 is disabled. 1: The period and compare registers of timer T13 are loaded from their shadow latches when T13 reaches the respective period value. *Note: STE13 is cleared by hardware after the shadow latch transfer.*
ECT13O	**Enable compare timer T13 output** 0: When ECT13O is cleared and timer T13 is running, signal COUT63 outputs the corresponding port latch value. 1: When ECT13O is set and timer T13 is running, timer T13 output COUT63 is enabled and outputs the PWM signal of the 10-bit compare channel.
CT13P	**Timer T13 Period Flag** The period flag CT13P is set whenever the contents of timer T13 match the contents of the timer T13 period register. This also generates an interrupt request. Bit CT13P must be cleared by software.

TRCON

Trap Enable Register **SFR (FF34$_H$/9A$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRP EN	**TRF**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
rw	rwh	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Bit	Function
TRF	**Trap Flag** TRF is set by hardware if the trap function is enabled (TRPEN=1) and $\overline{\text{CTRAP}}$ becomes active (low). If enabled, an interrupt is generated when TRF is set. TRF must be cleared by software.
TRPEN	**External $\overline{\text{CTRAP}}$ Trap Function Enable Bit** 0: External trap input $\overline{\text{CTRAP}}$ is disabled (default after reset). 1: External trap input $\overline{\text{CTRAP}}$ is enabled.

CC6MCON
CAPCOM6 Mode Ctrl. Reg. **SFR ($FF32_H$/99_H)** **Reset value: $00FF_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BC MP BC EM	**MPWM**		**EB CE**	**BC ERR**	**BC EN**	**BCM**		**COUT 3I**	**COUT XI**	**COUT 2I**	**CC2I**	**COUT 1I**	**CC1I**	**COUT 0I**	**CC0I**
rw	rw		rw	rwh	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
CCnI	**Compare Output CC6n Initial Value** (n = 0...2) The compare output CC6n drives the value of CCnI when the compare timer T12 is not running. CCnI represents the passive output level for an enabled compare channel. *Note: The initial values are only valid for capture/compare outputs, which are enabled for compare mode operation (compare output).*
COUTnI	**Compare Output COUT6n Initial Value** (n = 0...2) The compare output COUT6n drives the value of COUTnI when the compare timer T12 is not running. COUTnI represents the passive output level for an enabled compare channel. *Note: The initial values are only valid for capture/compare outputs, which are enabled for compare mode operation (compare output).*
COUTXI	**COUT6n Inversion Control** 0: T13's output signal is directly connected to compare outputs COUT6n in burst or multi-channel mode (n=0...2). 1: T13's output signal is inverted and then connected to compare outputs COUT6n in burst or multi-channel mode (n=0...2).
COUT3I	**Compare Output COUT63 Initial Value** This bit defines the initial logic state of the output COUT63 before timer T13 is started the first time. Further, COUT3I defines the logic state of COUT63 when bit ECT13O is reset (COUT63 disabled).
BCM	**Multi-channel PWM Mode Output Pattern Selection** This bitfield selects the output signal pattern in all multi-channel PWM modes (also refer to bitfield MPWM). 00: Idle mode. 01: Rotate right mode. 10: Rotate left mode. 11: Slow down mode.

Bit	Function
BCEN	**Block Commutation Enable** 0: The multi-channel PWM modes of the 16-bit capture/compare channels (selected by bitfield MPWM) are disabled. 1: The multi-channel PWM modes are enabled. *Note: Before bit BCEN is set, all required PWM compare outputs should be programmed to operate as compare outputs by writing to register CC6MSEL.*
BCERR	**Block Commutation Mode Error Flag** 0: No error condition. 1: An error condition in rotate right or rotate left mode has occurred: - After a transition at $\overline{\text{CC6POSx}}$ all $\overline{\text{CC6POSx}}$ inputs are at high or low level. - A “wrong follower“ condition has occurred (see description of bit BCEM). If the block commutation interrupt is enabled (EBCE=’1’) also a CAPCOM6 emergency interrupt will be generated. BCERR must be cleared by software.
EBCE	**Enable Block Commutation Mode Error Interrupt** 0: Block commutation mode error does not generate an interrupt. 1: The emergency interrupt is activated for a block commutation mode error. Refer to the description of bits BCERR and BCEM.
MPWM	**Multi-channel PWM Mode Selection** This bitfield selects the output signal pattern in all multi-channel PWM modes (also refer to bitfield BCM). 00: 3-phase block commutation mode. 01: 4-phase multi-channel PWM mode. 10: 5-phase multi-channel PWM mode. 11: 6-phase multi-channel PWM mode.

Bit	Function
BCMP	**Machine polarity** (Valid only in multi-channel PWM mode) 0: Only the COUT6n outputs are switched to the timer T13 output signal during the active phase in multi-channel PWM mode. CMSELn3 must be set for that functionality. 1: All enabled compare outputs COUT6n and CC6n are switched to the timer T13 output signal during their active phase in multi-channel PWM mode.
BCEM	**Error mode select bit** (Valid only in block commutation mode) 0: A “wrong follower“ condition is not notified as an error. 1: A “wrong follower“ condition in rotate right or rotate left mode sets flag BCERR if EBCE is set.

Note: When a multi-channel PWM mode is initiated the first time after reset, CC6MCON must be written twice: first write operation with bit BCEN cleared and all other bits set/cleared as required (BCM must be ’00’ for idle mode), followed by a second write operation with the same CC6MCON bit pattern of the first write operation but with BCEN set. After this second CC6MCON write operation, timer T12 can be started (setting CT12R in CTCON) and thereafter BCM can be put into another mode than the idle mode.

CC6MSEL
CAPCOM6 Mode Select Reg. ESFR (F036$_H$/1B$_H$) Reset value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ESMC	NMCS	-	-	CMSEL23	CMSEL2			CMSEL13	CMSEL1			CMSEL03	CMSEL0		
rw	rw	-	-	rw	rw			rw	rw			rw	rw		

Bit	Function
CMSELn	**Capture/Compare Mode Selection** These bitfields select/enable the operating mode and the output/input pin configuration of the 16-bit capture/compare channels. Each channel can be programmed individually either for compare or capture operation. 000: Compare outputs disabled, CC6n/COUT6n can be used for IO. 001: Compare output on pin CC6n, COUT6n can be used for IO. 010: Compare output on pin COUT6n, CC6n can be used for IO. 011: Compare output on pins COUT6n and CC6n. 100: Capture mode, not triggered by CC6n. COUT6n is IO. 101: Capture mode, trigg'd by a rising edge on CC6n. COUT6n is IO. 110: Capture mode, trigg'd by a falling edge on CC6n. COUT6n is IO. 111: Capture mode, trigg'd by any transition on CC6n. COUT6n is IO.
CMSELn3	**COUT6n Control by Timer T13 in Compare Mode** This bit determines if the output COUT6n is modulated during its active phase (defined via register CC6MCON) by the output signal of the 10-bit compare channel, typically a higher frequency signal. 0: COUT6n drives its active level. 1: COUT6n is modulated by the output signal of the 10-bit compare channel.
NMCS	**Next Multi-Channel PWM State** (Valid when ESMC = '1') 0: Idle. 1: Select the next follower state in the 4/5/6-phase Multi-Channel PWM modes. NMCS is reset by hardware in the next clock cycle after it has been set.
ESMC	**Enable Software Controlled Multi-Channel PWM Modes** Defines the follower state selection in the 4/5/6-phase multi-channel PWM modes. 0: Follower state selection controlled by compare timer T12. 1: Follower state selection controlled by bit NMCS (software control).

CC6MIC **SFR (FF36$_H$/9B$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CT12 FP	CT12 FC	CC2 F	CC2 R	CC1 F	CC1 R	CC0 R	CC0 R	EC TP	EC TC	CC2 FEN	CC2 REN	CC1 FEN	CC1 REN	CC0 FEN	CC0 REN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

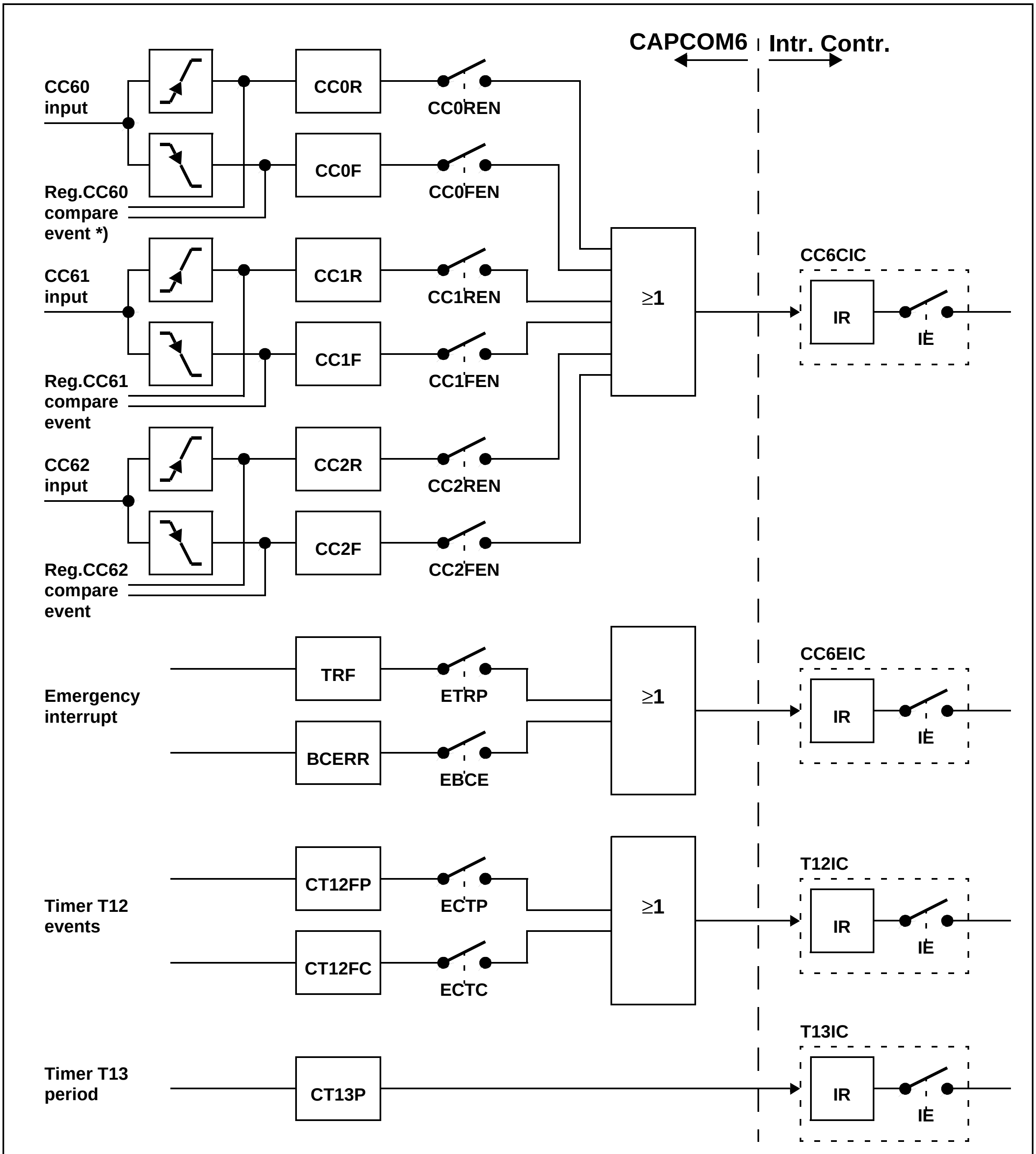
Bit	Function
CCnREN	**Capture/Compare Rising Edge Interrupt Enable** 0: Rising edge interrupt disabled. 1: An interrupt from request flag CCnR is enabled.
CCnFEN	**Capture/Compare Falling Edge Interrupt Enable** 0: Falling edge interrupt disabled. 1: An interrupt from request flag CCnF is enabled.
ECTC	**Enable Timer T12 Count Direction Change Interrupt** 0: Count direction change interrupt disabled. 1: An interrupt from request flag CT12FC is enabled **Note**: No effect in edge aligned mode.
ECTP	**Enable Timer T12 Period Interrupt** 0: Period interrupt disabled. 1: An interrupt from request flag CT12FP is enabled.
CCnR	**Capture/Compare Rising Edge Interrupt Flag** 0: Idle. 1: The interrupt request flag is set.. . . ... in **capture mode**: upon a rising edge at the corresponding CC6n inp. ... in **compare mode**: when T12 matches compare register CC6n while counting up (in both operating modes of timer T12).
CCnF	**Capture/Compare Falling Edge Interrupt Flag** 0: Idle. 1: The interrupt request flag is set.. . . ... in **capture mode**: upon a falling edge at the corresponding CC6n inp. ... in **compare mode**: when T12 matches compare register CC6n while counting down (in center aligned mode, timer T12 only).

Bit	Function
CT12FC	**Timer T12 Count Direction Change Flag** 0: Idle. 1: An interrupt request is generated when T12 (counting down in center aligned mode) matches 0000_H and changes to counting up. There is no effect in edge aligned mode.
CT12FP	**Timer T12 Period Flag** 0: Idle. 1: An interrupt request is generated when T12 matches the period value.

Note: All CAPCOM6 interrupt request bits in register CC6MIC must be cleared by software.

The CAPCOM6 Interrupt Structure

The figure below summarizes the CAPCOM6's interrupt sources and the related status and control flags, and shows the association with the 4 CAPCOM6 interrupt nodes.



*) Internal compare event in case of block commutation mode.

Figure 17-13 CAPCOM6 Interrupt Structure

Interrupt Node Control Registers

T12IC **ESFR (F190$_H$/C8$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T12 IR	**T12 IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

T13IC **ESFR (F198$_H$/CC$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T13 IR	**T13 IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

CC6EIC **ESFR (F188$_H$/C4$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								CC6 EIR	**CC6 EIE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

CC6CIC **ESFR (F17E$_H$/BF$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								CC6 IR	**CC6 EI**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

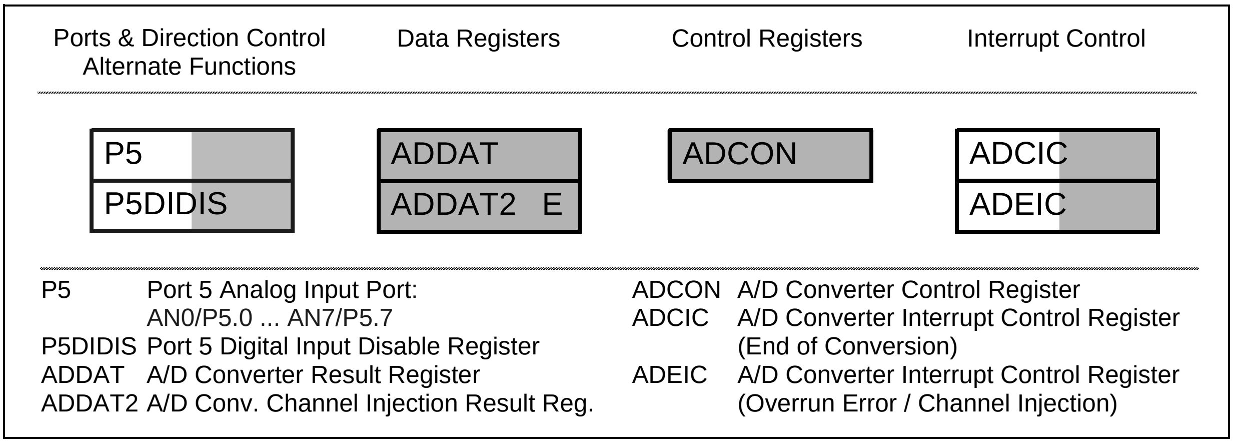
18 The Analog / Digital Converter

The C164 provides an Analog / Digital Converter with 10-bit resolution and a sample & hold circuit on-chip. A multiplexer selects between up to 8 analog input channels (alternate functions of Port 5) either via software (fixed channel modes) or automatically (auto scan modes).

To fulfill most requirements of embedded control applications the ADC supports the following conversion modes:

- **Fixed Channel Single Conversion**
  produces just one result from the selected channel
- **Fixed Channel Continuous Conversion**
  repeatedly converts the selected channel
- **Auto Scan Single Conversion**
  produces one result from each of a selected group of channels
- **Auto Scan Continuous Conversion**
  repeatedly converts the selected group of channels
- **Wait for ADDAT Read Mode**
  start a conversion automatically when the previous result was read
- **Channel Injection Mode**
  insert the conversion of a specific channel into a group conversion (auto scan)

A set of SFRs and port pins provide access to control functions and results of the ADC.



P5	Port 5 Analog Input Port: AN0/P5.0 ... AN7/P5.7	ADCON	A/D Converter Control Register
P5DIDIS	Port 5 Digital Input Disable Register	ADCIC	A/D Converter Interrupt Control Register (End of Conversion)
ADDAT	A/D Converter Result Register	ADEIC	A/D Converter Interrupt Control Register (Overrun Error / Channel Injection)
ADDAT2	A/D Conv. Channel Injection Result Reg.		

Figure 18-1 SFRs and Port Pins associated with the A/D Converter

The external analog reference voltages V_{AREF} and V_{AGND} are fixed. The separate supply for the ADC reduces the interference with other digital signals.

The sample time as well as the conversion time is programmable, so the ADC can be adjusted to the internal resistances of the analog sources and/or the analog reference voltage supply.

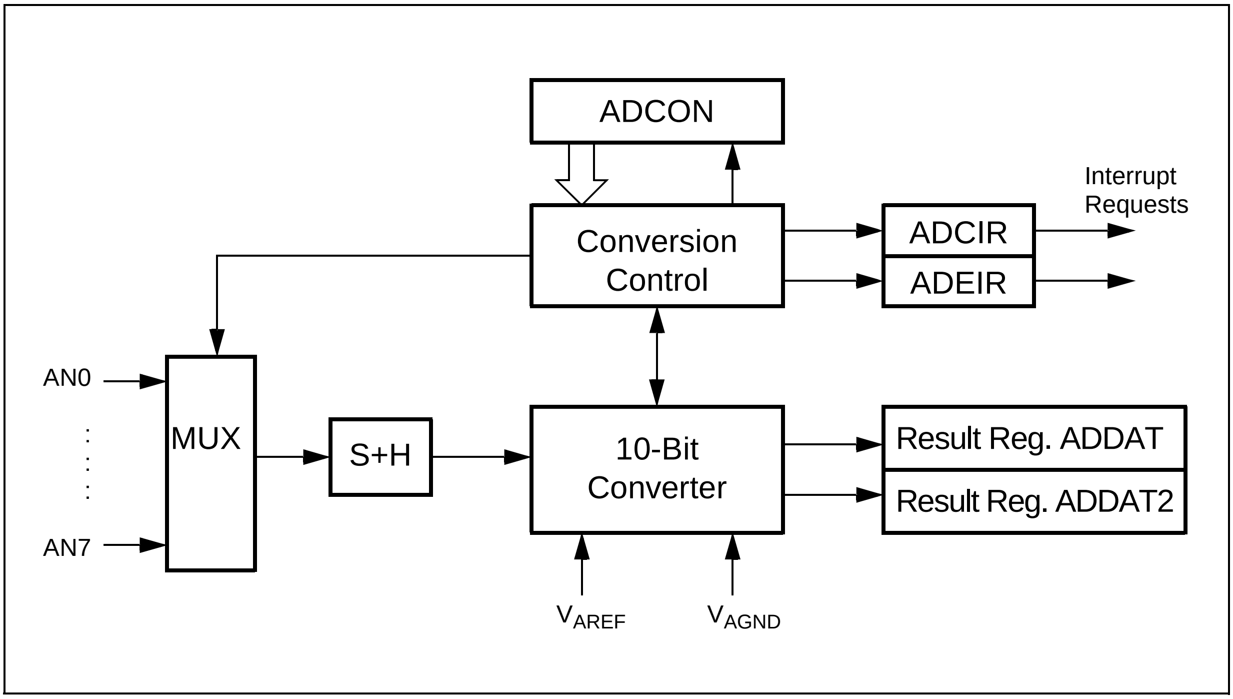


Figure 18-2 Analog / Digital Converter Block Diagram

18.1 Mode Selection and Operation

The analog input channels AN7...AN0 are alternate functions of Port 5 which is an input-only port. The Port 5 lines may either be used as analog or digital inputs. For pins that shall be used as analog inputs it is recommended to disable the digital input stage via register P5DIDIS. This avoids undesired cross currents and switching noise while the (analog) input signal level is between V_{IL} and V_{IH}.

The functions of the A/D converter are controlled by the bit-addressable A/D Converter Control Register ADCON. Its bitfields specify the analog channel to be acted upon, the conversion mode, and also reflect the status of the converter.

ADCON
ADC Control Register **SFR (FFA0$_H$/D0$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADCTC		**ADSTC**		**AD CRQ**	**AD CIN**	**AD WR**	**AD BSY**	**AD ST**	**-**	**ADM**		**ADCH**			
rw		rw		rwh	rw	rw	rwh	rwh	-	rw		rw			

Bit	Function
ADCH	**ADC Analog Channel Input Selection** Selects the (first) ADC channel which is to be converted. *Note: Valid channel numbers are 0$_H$ to 7$_H$.*
ADM	**ADC Mode Selection** 00: Fixed Channel Single Conversion 01: Fixed Channel Continuous Conversion 10: Auto Scan Single Conversion 11: Auto Scan Continuous Conversion
ADST	**ADC Start Bit** 0: Stop a running conversion 1: Start conversion(s)
ADBSY	**ADC Busy Flag** 0: ADC is idle 1: A conversion is active.
ADWR	**ADC Wait for Read Control**
ADCIN	**ADC Channel Injection Enable**
ADCRQ	**ADC Channel Injection Request Flag**
ADSTC	**ADC Sample Time Control** (Defines the ADC sample time in a certain range) 00: t_{BC} * 8 01: t_{BC} * 16 10: t_{BC} * 32 11: t_{BC} * 64
ADCTC	**ADC Conversion Time Control** (Defines the ADC basic conversion clock f_{BC}) 00: $f_{BC} = f_{CPU} / 4$ 01: $f_{BC} = f_{CPU} / 2$ 10: $f_{BC} = f_{CPU} / 16$ 11: $f_{BC} = f_{CPU} / 8$

Bitfield ADCH specifies the analog input channel which is to be converted (first channel of a conversion sequence in auto scan modes). Bitfield ADM selects the operating mode of the A/D converter. A conversion (or a sequence) is then started by setting bit ADST. Clearing ADST stops the A/D converter after a certain operation which depends on the selected operating mode.

The busy flag (read-only) ADBSY is set, as long as a conversion is in progress.

The result of a conversion is stored in the result register ADDAT, or in register ADDAT2 for an injected conversion.

Note: Bitfield CHNR of register ADDAT is loaded by the ADC to indicate, which channel the result refers to.
Bitfield CHNR of register ADDAT2 is loaded by the CPU to select the analog channel, which is to be injected.

ADDAT
ADC Result Register **SFR (FEA0$_H$/50$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHNR				-	-	ADRES									
rwh				-	-	rwh									

ADDAT2
ADC Chan. Inj. Result Reg. **ESFR (F0A0$_H$/50$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHNR				-	-	ADRES									
rw				-	-	rwh									

Bit	Function
ADRES	**A/D Conversion Result** The 10-bit digital result of the most recent conversion.
CHNR	**Channel Number** (identifies the converted analog channel) *Note: Valid channel numbers are 7$_H$ to 0$_H$.*

A conversion is started by setting bit ADST=‘1’. The busy flag ADBSY will be set and the converter then selects and samples the input channel, which is specified by the channel selection field ADCH in register ADCON. The sampled level will then be held internally during the conversion. When the conversion of this channel is complete, the 10-bit result together with the number of the converted channel is transferred into the result register ADDAT and the interrupt request flag ADCIR is set. The conversion result is placed into bitfield ADRES of register ADDAT.

If bit ADST is reset via software, while a conversion is in progress, the A/D converter will stop after the current conversion (fixed channel modes) or after the current conversion sequence (auto scan modes).

Setting bit ADST while a conversion is running, will abort this conversion and start a new conversion with the parameters specified in ADCON.

Note: Abortion and restart (see above) are triggered by bit ADST changing from ‘0’ to ‘1’, i.e. ADST must be ‘0’ before being set.

While a conversion is in progress, the mode selection field ADM and the channel selection field ADCH may be changed. ADM will be evaluated after the current conversion. ADCH will be evaluated after the current conversion (fixed channel modes) or after the current conversion sequence (auto scan modes).

Fixed Channel Conversion Modes

These modes are selected by programming the mode selection bitfield ADM in register ADCON to ‘00_B’ (single conversion) or to ‘01_B’ (continuous conversion). After starting the converter through bit ADST the busy flag ADBSY will be set and the channel specified in bit field ADCH will be converted. After the conversion is complete, the interrupt request flag ADCIR will be set.

In Single Conversion Mode the converter will automatically stop and reset bits ADBSY and ADST.

In Continuous Conversion Mode the converter will automatically start a new conversion of the channel specified in ADCH. ADCIR will be set after each completed conversion.

When bit ADST is reset by software, while a conversion is in progress, the converter will complete the current conversion and then stop and reset bit ADBSY.

Auto Scan Conversion Modes

These modes are selected by programming the mode selection field ADM in register ADCON to ‘10_B’ (single conversion) or to ‘11_B’ (continuous conversion). Auto Scan modes automatically convert a sequence of analog channels, beginning with the channel specified in bit field ADCH and ending with channel 0, without requiring software to change the channel number.
After starting the converter through bit ADST, the busy flag ADBSY will be set and the channel specified in bit field ADCH will be converted. After the conversion is complete, the interrupt request flag ADCIR will be set and the converter will automatically start a new conversion of the next lower channel. ADCIR will be set after each completed conversion. After conversion of channel 0 the current sequence is complete.

In Single Conversion Mode the converter will automatically stop and reset bits ADBSY and ADST.

In Continuous Conversion Mode the converter will automatically start a new sequence beginning with the conversion of the channel specified in ADCH.

When bit ADST is reset by software, while a conversion is in progress, the converter will complete the current sequence (including conversion of channel 0) and then stop and reset bit ADBSY.

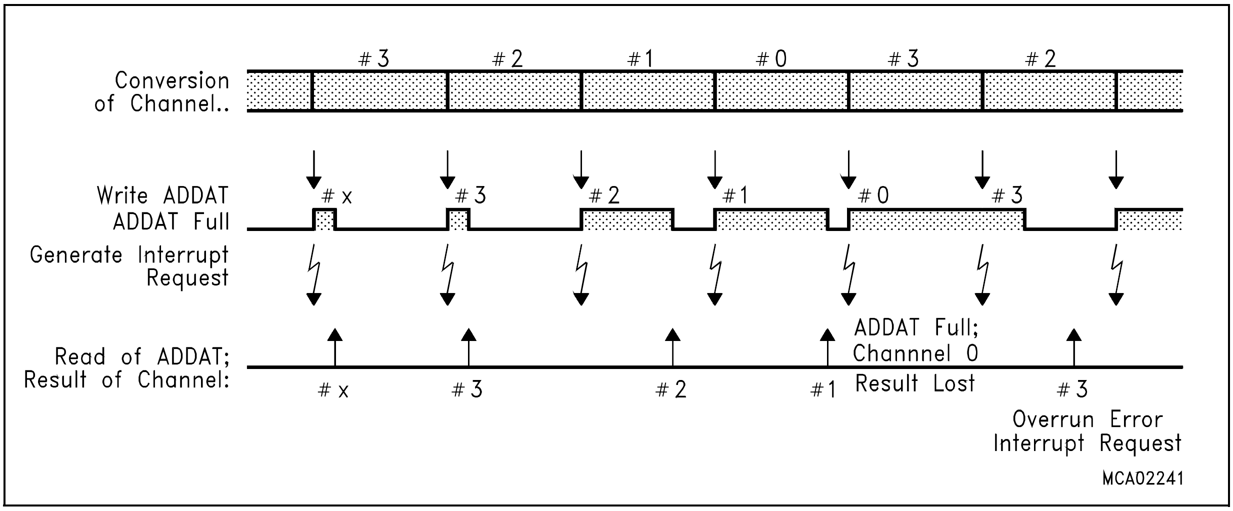

Conversion of Channel..
# 3
# 2
# 1
# 0
# 3
# 2
Write ADDAT
ADDAT Full
# x
# 3
# 2
# 1
# 0
# 3
Generate Interrupt Request
Read of ADDAT; Result of Channel:
# x
# 3
# 2
# 1
ADDAT Full; Channnel 0 Result Lost
# 3
Overrun Error Interrupt Request
MCA02241

Figure 18-3 Auto Scan Conversion Mode Example

Wait for ADDAT Read Mode

If in default mode of the ADC a previous conversion result has not been read out of register ADDAT by the time a new conversion is complete, the previous result in register ADDAT is lost because it is overwritten by the new value, and the A/D overrun error interrupt request flag ADEIR will be set.

In order to avoid error interrupts and the loss of conversion results especially when using continuous conversion modes, the ADC can be switched to "Wait for ADDAT Read Mode" by setting bit ADWR in register ADCON.

If the value in ADDAT has not been read by the time the current conversion is complete, the new result is stored in a temporary buffer and the next conversion is suspended (ADST and ADBSY will remain set in the meantime, but no end-of-conversion interrupt will be generated). After reading the previous value from ADDAT the temporary buffer is copied into ADDAT (generating an ADCIR interrupt) and the suspended conversion is started. This mechanism applies to both single and continuous conversion modes.

Note: While in standard mode continuous conversions are executed at a fixed rate (determined by the conversion time), in "Wait for ADDAT Read Mode" there may be delays due to suspended conversions. However, this only affects the conversions, if the CPU (or PEC) cannot keep track with the conversion rate.

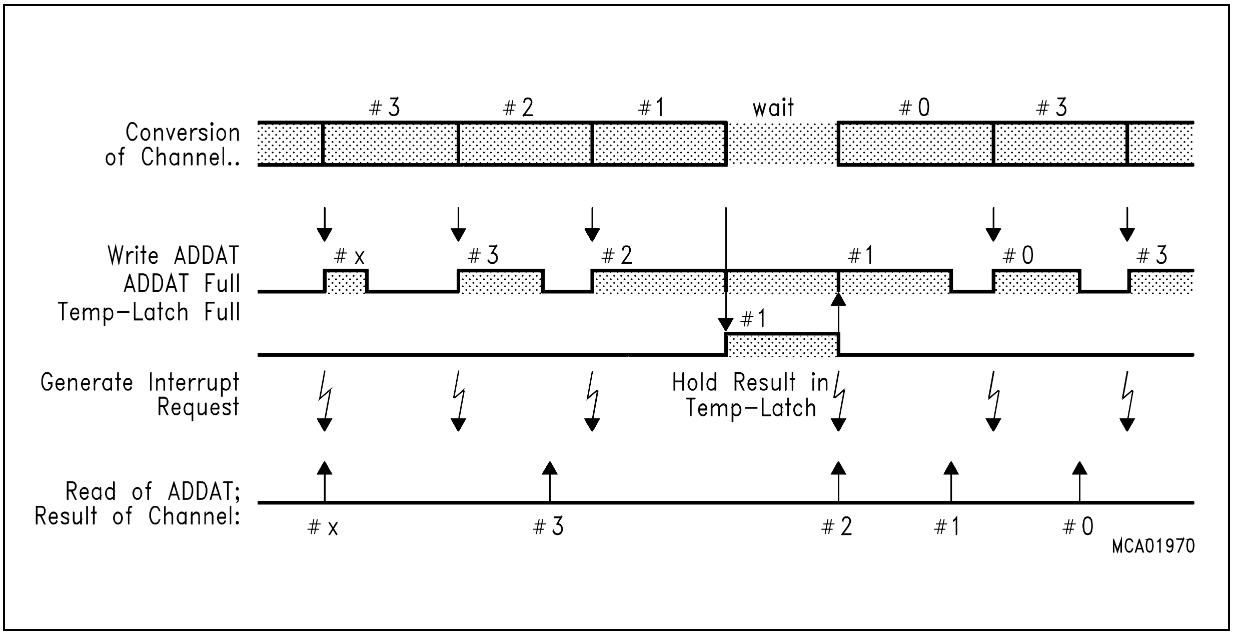


Figure 18-4 Wait for Read Mode Example

Channel Injection Mode

Channel Injection Mode allows the conversion of a specific analog channel (also while the ADC is running in a continuous or auto scan mode) without changing the current operating mode. After the conversion of this specific channel the ADC continues with the original operating mode.

Channel Injection mode is enabled by setting bit ADCIN in register ADCON and requires the Wait for ADDAT Read Mode (ADWR=‘1’). The channel to be converted in this mode is specified in bitfield CHNR of register ADDAT2.

Note: Bitfield CHNR in ADDAT2 is not modified by the A/D converter, but only the ADRES bit field. Since the channel number for an injected conversion is not buffered, bitfield CHNR of ADDAT2 must never be modified during the sample phase of an injected conversion, otherwise the input multiplexer will switch to the new channel. It is recommended to only change the channel number with no injected conversion running.

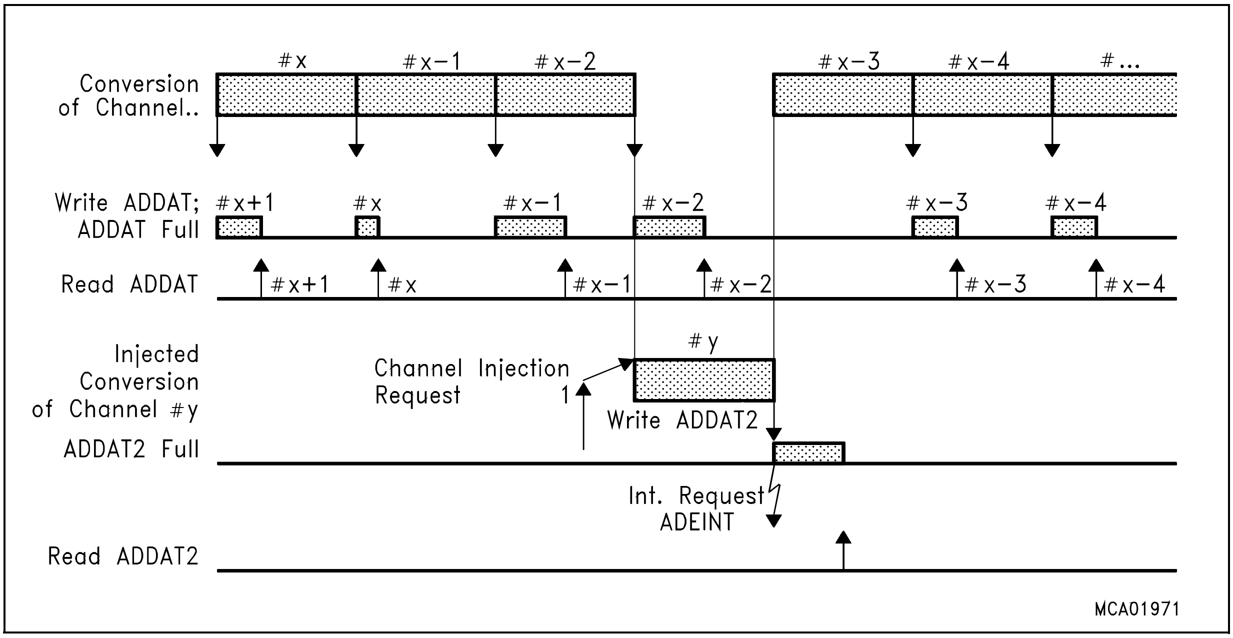


Figure 18-5 Channel Injection Example

A channel injection can be triggered in two ways:

- setting of the Channel Injection Request bit ADCRQ via software
- a compare or a capture event of Capture/Compare register CC27 of the CAPCOM2 unit, which also sets bit ADCRQ.

The second method triggers a channel injection at a specific time, on the occurrence of a predefined count value of the CAPCOM timers or on a capture event of register CC27. This can be either the positive, the negative, or both the positive and the negative edge of an external signal. In addition, this option allows recording the time of occurrence of this signal.

Note: The channel injection request bit ADCRQ will be set on any interrupt request of CAPCOM2 channel CC27, regardless whether the channel injection mode is enabled or not. It is recommended to always clear bit ADCRQ before enabling the channel injection mode.

After the completion of the current conversion (if any is in progress) the converter will start (inject) the conversion of the specified channel. When the conversion of this channel is complete, the result will be placed into the alternate result register ADDAT2, and a Channel Injection Complete Interrupt request will be generated, which uses the interrupt request flag ADEIR (for this reason the Wait for ADDAT Read Mode is required).

Note: If the temporary data register used in Wait for ADDAT Read Mode is full, the respective next conversion (standard or injected) will be suspended. The temporary register can hold data for ADDAT (from a standard conversion) or for ADDAT2 (from an injected conversion).

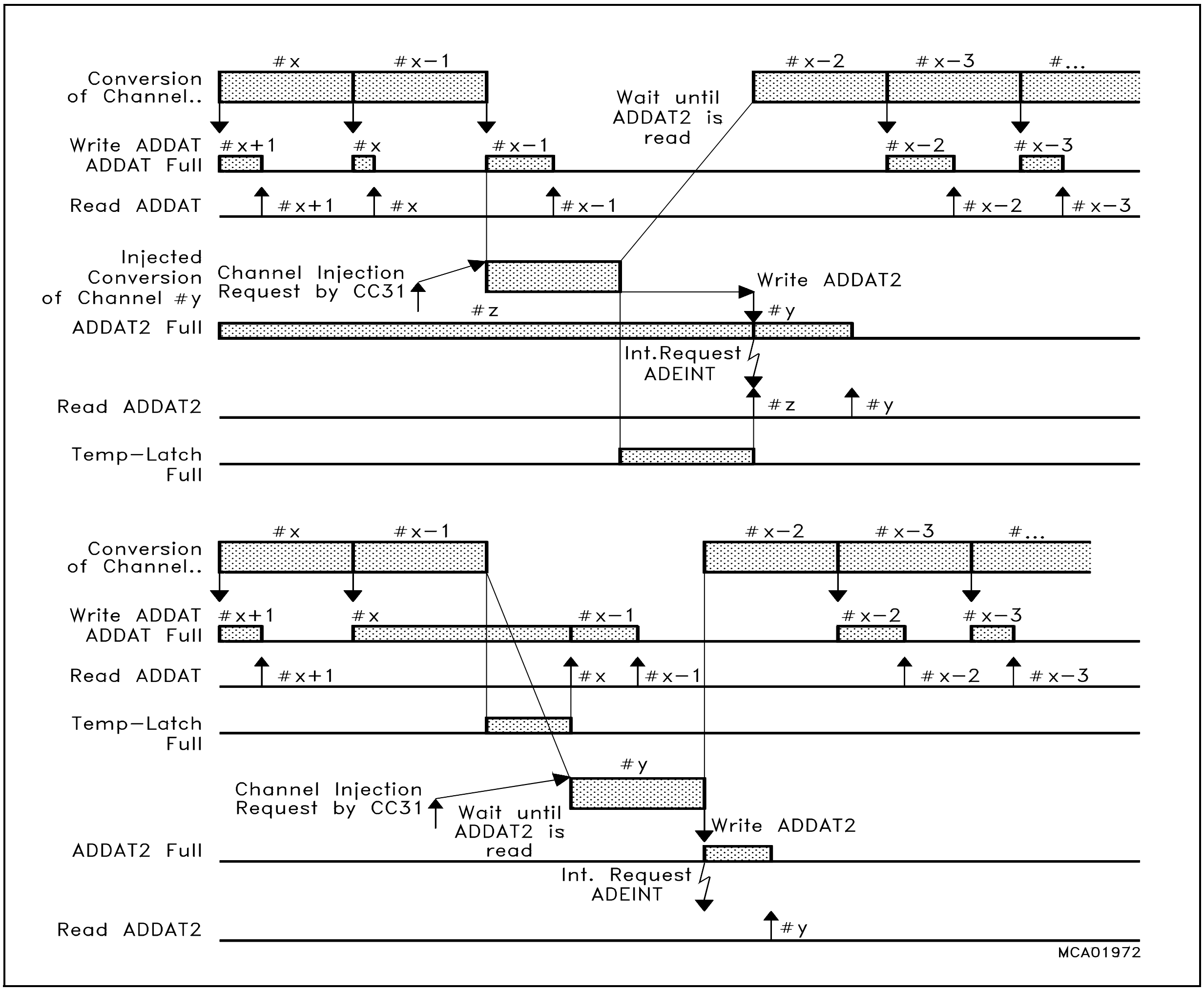


Figure 18-6 Channel Injection Example with Wait for Read

Arbitration of Conversions

Conversion requests that are activated while the ADC is idle immediately trigger the respective conversion. If a conversion is requested while another conversion is currently in progress the operation of the A/D converter depends on the kind of the involved conversions (standard or injected).

Note: A conversion request is activated if the respective control bit (ADST or ADCRQ) is toggled from ’0’ to ’1’, i.e. the bit must have been zero before being set.

The table below summarizes the ADC operation in the possible situations.

Table 18-1 Conversion Arbitration

Conversion in progress	**New requested conversion**	
	Standard	**Injected**
Standard	Abort running conversion, and start requested new conversion.	Complete running conversion, start requested conversion after that.
Injected	Complete running conversion, start requested conversion after that.	Complete running conversion, start requested conversion after that. Bit ADCRQ will be ’0’ for the second conversion, however.

18.2 Conversion Timing Control

When a conversion is started, first the capacitances of the converter are loaded via the respective analog input pin to the current analog input voltage. The time to load the capacitances is referred to as sample time. Next the sampled voltage is converted to a digital value in successive steps, which correspond to the resolution of the ADC. During these phases (except for the sample time) the internal capacitances are repeatedly charged and discharged via pins V_{AREF} and V_{AGND}.

The current that has to be drawn from the sources for sampling and changing charges depends on the time that each respective step takes, because the capacitors must reach their final voltage level within the given time, at least with a certain approximation. The maximum current, however, that a source can deliver, depends on its internal resistance.

The time that the two different actions during conversion take (sampling, and converting) can be programmed within a certain range in the C164 relative to the CPU clock. The absolute time that is consumed by the different conversion steps therefore is independent from the general speed of the controller. This allows adjusting the A/D converter of the C164 to the properties of the system:

Fast Conversion can be achieved by programming the respective times to their absolute possible minimum. This is preferable for scanning high frequency signals. The internal resistance of analog source and analog supply must be sufficiently low, however.

High Internal Resistance can be achieved by programming the respective times to a higher value, or the possible maximum. This is preferable when using analog sources and supply with a high internal resistance in order to keep the current as low as possible. The conversion rate in this case may be considerably lower, however.

The conversion time is programmed via the upper two bits of register ADCON. Bitfield ADCTC (conversion time control) selects the basic conversion clock (f_{BC}), used for the operation of the A/D converter. The sample time is derived from this conversion clock. The table below lists the possible combinations. The timings refer to CPU clock cycles, where $t_{CPU} = 1 / f_{CPU}$.

The limit values for f_{BC} (see data sheet) must not be exceeded when selecting ADCTC and f_{CPU}.

Table 18-2 ADC Conversion Timing Control

ADCON.15|14 (ADCTC)	A/D Converter Basic clock f_{BC}	ADCON.13|12 (ADSTC)	Sample time t_S
00	f_{CPU} / 4	00	t_{BC} * 8
01	f_{CPU} / 2	01	t_{BC} * 16
10	f_{CPU} / 16	10	t_{BC} * 32
11	f_{CPU} / 8	11	t_{BC} * 64

The time for a complete conversion includes the sample time t_{S}, the conversion itself and the time required to transfer the digital value to the result register (2 t_{CPU}) as shown in the example below.

Note: The non-linear decoding of bit field ADCTC provides compatibility with 80C166 designs for the default value (‘00’ after reset).

Converter Timing Example

Assumptions:	f_{CPU}	= 25 MHz (i.e. t_{CPU} = 40 ns), ADCTC = ’00’, ADSTC = ’00’.
Basic clock	f_{BC}	= f_{CPU} / 4 = 6.25 MHz, i.e. t_{BC} = 160 ns.
Sample time	t_{S}	= t_{BC} * 8 = 1280 ns.
Conversion time	t_{C}	= t_{S} + 40 t_{BC} + 2 t_{CPU} = (1280 + 6400 + 80) ns = 7.76 µs.

Note: For the exact specification please refer to the data sheet of the selected derivative.

18.3 A/D Converter Interrupt Control

At the end of each conversion, interrupt request flag ADCIR in interrupt control register ADCIC is set. This end-of-conversion interrupt request may cause an interrupt to vector ADCINT, or it may trigger a PEC data transfer which reads the conversion result from register ADDAT e.g. to store it into a table in the internal RAM for later evaluation.

The interrupt request flag ADEIR in register ADEIC will be set either, if a conversion result overwrites a previous value in register ADDAT (error interrupt in standard mode), or if the result of an injected conversion has been stored into ADDAT2 (end-of-injected-conversion interrupt). This interrupt request may be used to cause an interrupt to vector ADEINT, or it may trigger a PEC data transfer.

ADCIC
ADC Conversion Intr.Ctrl.Reg. SFR (FF98$_H$/CC$_H$) Reset value: - - 00$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								ADC IR	ADC IE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

ADEIC
ADC Error Intr.Ctrl.Reg. SFR (FF9A$_H$/CD$_H$) Reset value: - - 00$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								ADE IR	ADE IE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

19 The On-Chip CAN Interface

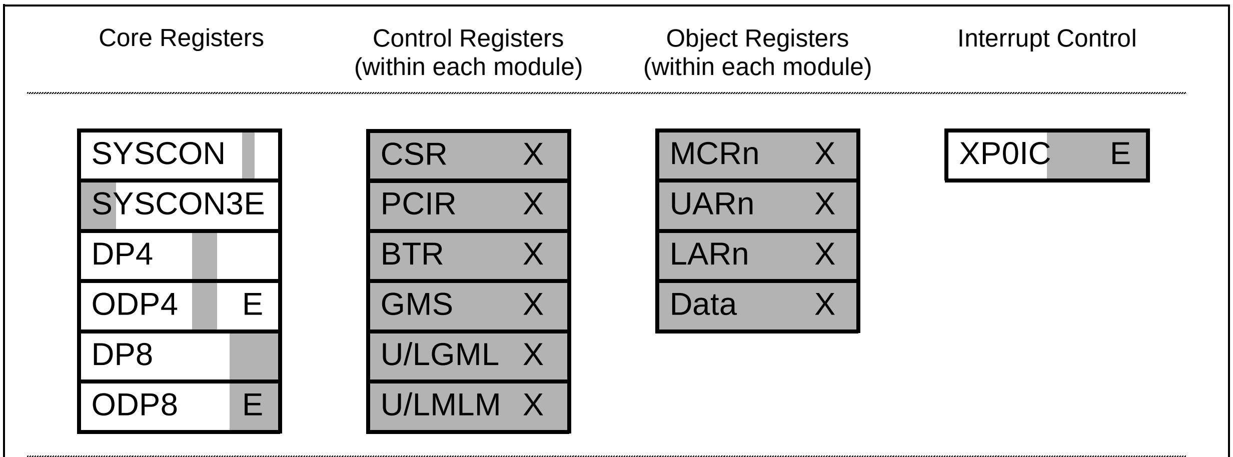
The Controller Area Network (CAN) bus with its associated protocol allows communication between a number of stations which are connected to this bus with high efficiency.
Efficiency in this context means:

- Transfer speed (Data rates of up to 1 Mbit/sec can be achieved)
- Data integrity (The CAN protocol provides several means for error checking)
- Host processor unloading (The controller here handles most of the tasks autonomously)
- Flexible and powerful message passing (The extended CAN protocol is supported)

The integrated CAN module handles the completely autonomous transmission and reception of CAN frames in accordance with the CAN specification V2.0 part B (active), i.e. the on-chip CAN module can receive and transmit...

- **Standard frames** with 11-bit identifiers, as well as
- **Extended frames** with 29-bit identifiers.

Note: The CAN module is an XBUS peripheral and therefore requires bit XPEN in register SYSCON to be set in order to be operable.



SYSCON	System Configuration Register	CSR	Control/Status Register
SYSCON3	Peripheral Management Control Register	PCIR	Port Control / Interrupt Register
DP4	Port 4 Direction Control Register	BTR	Bit Timing Register
ODP4	Port 4 Open Drain Control Register	GMS	Global Mask Short
DP8	Port 8 Direction Control Register	U/LGML	Global Mask Long
ODP8	Port 8 Open Drain Control Register	U/LMLM	Last Message Mask
XP0IC	CAN1 Interrupt Control Register	MCRn	Configuration Register of Message n
		U/LARn	Arbitration Register of Message n

Figure 19-1 Registers Associated with the CAN Module

The bit timing is derived from the XCLK and is programmable up to a data rate of 1 MBaud. The minimum CPU clock frequency to achieve 1 MBaud is $f_{CPU} \geq 8/16$ MHz, depending on the activation of the CAN module's clock prescaler.

The CAN module uses two pins of Port 4 or Port 8 to interface to a bus transceiver.

It provides **Full CAN** functionality on up to 15 full sized message objects (8 data bytes each). Message object 15 may be configured for **Basic CAN** functionality with a double-buffered receive object.

Both modes provide separate masks for acceptance filtering which allows the acceptance of a number of identifiers in Full CAN mode and also allows disregarding a number of identifiers in Basic CAN mode.

All message objects can be updated independent from the other objects during operation of the module and are equipped with buffers for the maximum message length of 8 bytes.

19.1 Functional Blocks of the CAN Module

The CAN module combines several functional blocks (see figure below) that work in parallel and contribute to the controller's performance. These units and the functions they provide are described below.

Each of the message objects has a unique identifier and its own set of control and status bits. Each object can be configured with its direction as either transmit or receive, except the last message which is only a double receive buffer with a special mask register.

An object with its direction set as transmit can be configured to be automatically sent whenever a remote frame with a matching identifier (taking into account the respective global mask register) is received over the CAN bus. By requesting the transmission of a message with the direction set as receive, a remote frame can be sent to request that the appropriate object be sent by some other node. Each object has separate transmit and receive interrupts and status bits, giving the CPU full flexibility in detecting when a remote/data frame has been sent or received.

For general purpose two masks for acceptance filtering can be programmed, one for identifiers of 11 bits and one for identifiers of 29 bits. However the CPU must configure bit XTD (Normal or Extended Frame Identifier) for each valid message to determine whether a standard or extended frame will be accepted.

The last message object has its own programmable mask for acceptance filtering, allowing a large number of infrequent objects to be handled by the system.

The object layer architecture of the CAN controller is designed to be as regular and orthogonal as possible. This makes it easy to use.

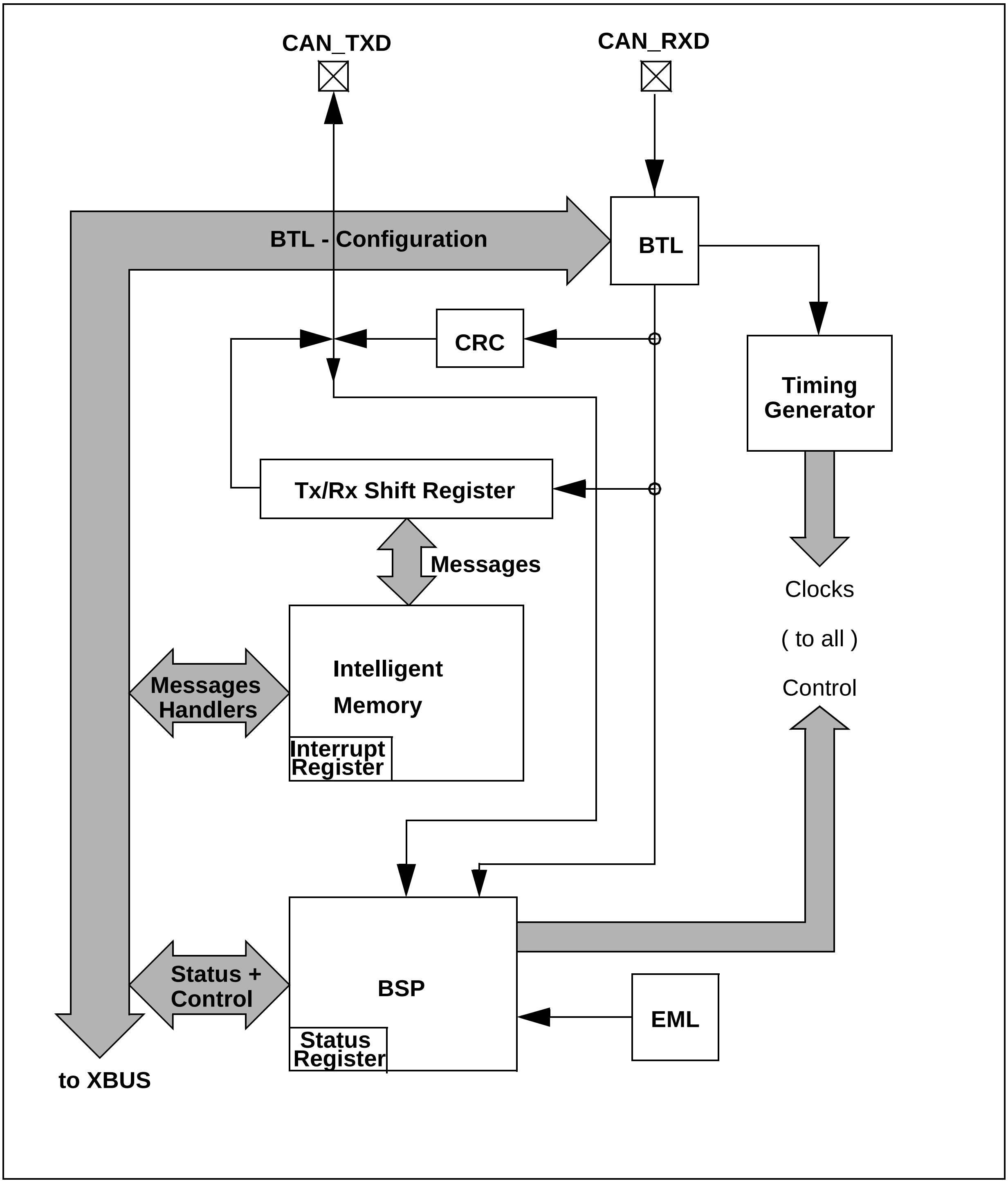


Figure 19-2 CAN Controller Block Diagram

Tx/Rx Shift Register

The Transmit / Receive Shift Register holds the destuffed bit stream from the bus line to allow the parallel access to the whole data or remote frame for the acceptance match test and the parallel transfer of the frame to and from the Intelligent Memory.

Bit Stream Processor

The Bit Stream Processor (BSP) is a sequencer controlling the sequential data stream between the Tx/Rx Shift Register, the CRC Register, and the bus line. The BSP also controls the EML and the parallel data stream between the Tx/Rx Shift Register and the Intelligent Memory such that the processes of reception, arbitration, transmission, and error signalling are performed according to the CAN protocol. Note that the automatic retransmission of messages which have been corrupted by noise or other external error conditions on the bus line is handled by the BSP.

Cyclic Redundancy Check Register

This register generates the Cyclic Redundancy Check (CRC) code to be transmitted after the data bytes and checks the CRC code of incoming messages. This is done by dividing the data stream by the code generator polynomial.

Error Management Logic

The Error Management Logic (EML) is responsible for the fault confinement of the CAN device. Its counters, the Receive Error Counter and the Transmit Error Counter, are incremented and decremented by commands from the Bit Stream Processor. According to the values of the error counters, the CAN controller is set into the states *error active*, *error passive* and *busoff*.

The CAN controller is *error active*, if both error counters are below the *error passive* limit of 128.

It is *error passive*, if at least one of the error counters equals or exceeds 128.

It goes *busoff*, if the Transmit Error Counter equals or exceeds the *busoff* limit of 256. The device remains in this state, until the *busoff* recovery sequence is finished.

Additionally, there is the bit EWRN in the Status Register, which is set, if at least one of the error counters equals or exceeds the error warning limit of 96. EWRN is reset, if both error counters are less than the error warning limit.

Bit Timing Logic

This block (BTL) monitors the busline input CAN_RXD and handles the busline related bit timing according to the CAN protocol.

The BTL synchronizes on a *recessive* to *dominant* busline transition at *Start of Frame* (hard synchronization) and on any further *recessive* to *dominant* busline transition, if the CAN controller itself does not transmit a *dominant* bit (resynchronization).

The BTL also provides programmable time segments to compensate for the propagation delay time and for phase shifts and to define the position of the *Sample Point* in the bit time. The programming of the BTL depends on the baudrate and on external physical delay times.

Intelligent Memory

The Intelligent Memory (CAM/RAM Array) provides storage for up to 15 message objects of maximum 8 data bytes length. Each of these objects has a unique identifier and its own set of control and status bits. After the initial configuration, the Intelligent Memory can handle the reception and transmission of data without further CPU actions.

Organization of Registers and Message Objects

All registers and message objects of the CAN controller are located in the special CAN address area of 256 bytes, which is mapped into segment 0 and uses addresses00’EF00$_H$ through 00’EFFF$_H$. All registers are organized as 16-bit registers, located on word addresses. However, all registers may be accessed bytewise in order to select special actions without effecting other mechanisms.

Register Naming reflects the specific name of a register as well as a general module indicator. This results in unique register names.

Example: module indicator is **C1** (CAN module 1), specific name is Control/Status Register (**CSR**), unique register name is **C1CSR**.

Note: The address map shown below lists the registers which are part of the CAN controller. There are also C164 specific registers that are associated with the CAN module.

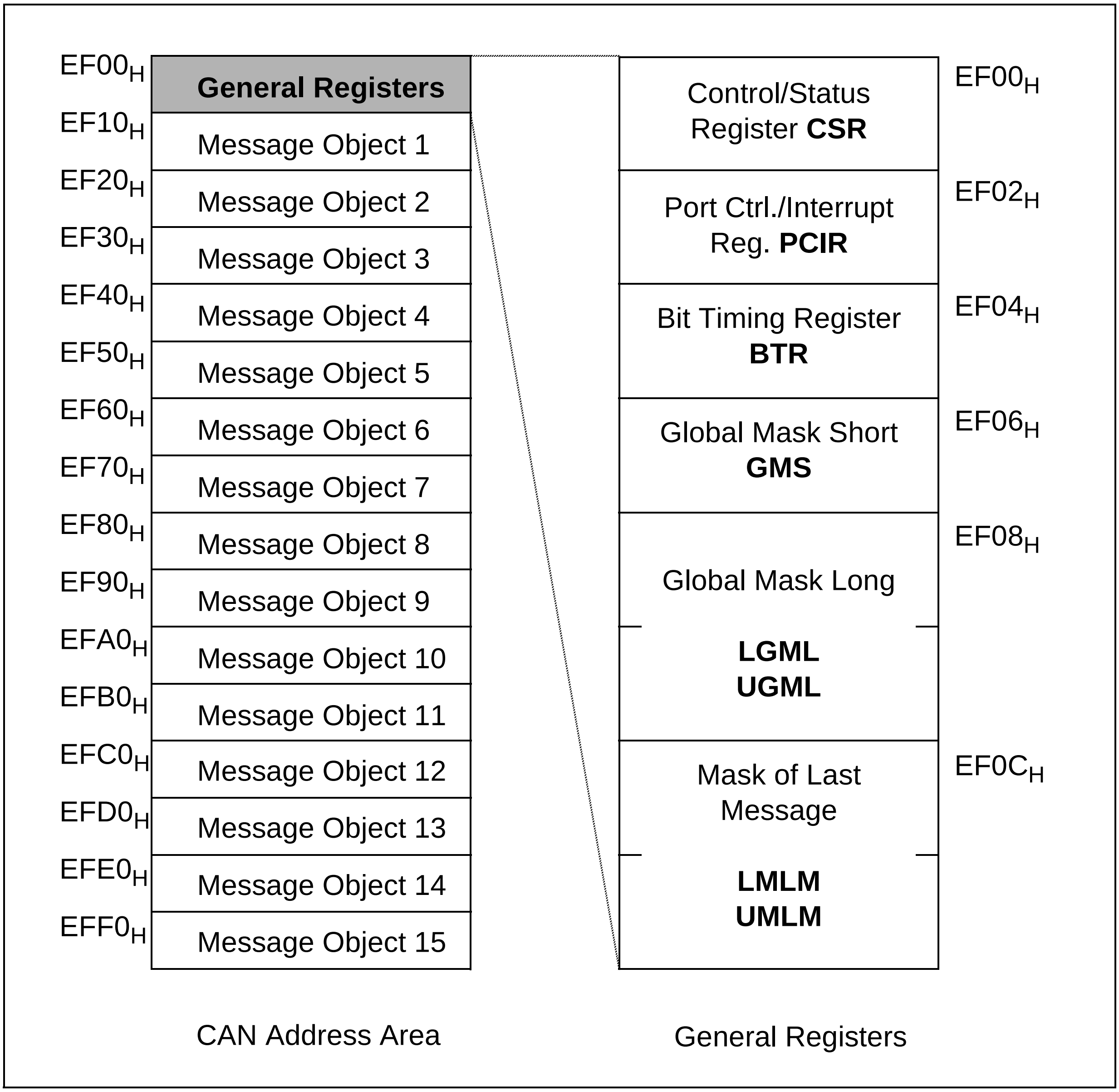


Figure 19-3 CAN Module Address Map

19.2 General Functional Description

The Control / Status Register (CSR) accepts general control settings for the module and provides general status information.

CSR
Control / Status Register **XReg (EF00$_H$)** **Reset value: XX01$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B OFF	E WRN	-	RX OK	TX OK	LEC			TM	CCE	0	CPS	EIE	SIE	IE	INIT
rh	rh	r	rwh	rwh	rwh			rw	rw	r	rw	rw	rw	rw	rwh

Bit	Function (Control Bits)
INIT	**Initialization** Starts the initialization of the CAN controller, when set. INIT is set -after a reset -when entering the *busoff* state -by the application software
IE	**Interrupt Enable** Enables or disables interrupt generation from the CAN module via the signal $\overline{\text{XINTR}}$. Does not affect status updates.
SIE	**Status Change Interrupt Enable** Enables or disables interrupt generation when a message transfer (reception or transmission) is successfully completed or a CAN bus error is detected (and registered in the status partition).
EIE	**Error Interrupt Enable** Enables or disables interrupt generation on a change of bit BOFF or EWARN in the status partition).
CPS	**Clock Prescaler Control Bit** 0: **Standard mode:** the input clock is divided 2:1. The minimum input frequency to achieve a baudrate of 1 MBaud is f_{CPU} = 16 MHz. 1: **Fast mode:** the input clock is used directly 1:1. The minimum input frequency to achieve a baudrate of 1 MBaud is f_{CPU} = 8 MHz.
CCE	**Configuration Change Enable** Allows or inhibits CPU access to the Bit Timing Register.
TM	**Test Mode** (must be '0') Make sure that this bit is always cleared when writing to the Control Register, as this bit controls a special test mode, that is used for production testing. During normal operation, however, this test mode may lead to undesired behaviour of the device.

Bit	Function (Status Bits)
LEC	**Last Error Code** This field holds a code which indicates the type of the last error occurred on the CAN bus. If a message has been transferred (reception or transmission) without error, this field will be cleared. 0 **No Error** 1 **Stuff Error:** More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed. 2 **Form Error:** Wrong format in fixed format part of a received frame. 3 **AckError:** The message this CAN controller transmitted was not acknowledged by another node. 4 **Bit1Error:** During the transmission of a message (with the exception of the arbitration field), the device wanted to send a *recessive* level (“1”), but the monitored bus value was *dominant*. 5 **Bit0Error:** During the transmission of a message (or acknowledge bit, active error flag, or overload flag), the device wanted to send a *dominant* level (“0”), but the monitored bus value was *recessive*. During *busoff* recovery this status is set each time a sequence of 11 *recessive* bits has been monitored. This enables the CPU to monitor the proceeding of the busoff recovery sequence (indicates that the bus is not stuck at *dominant* or continously disturbed). 6 **CRCError:** The received CRC check sum was incorrect. 7 **Unused code:** may be written by the CPU to check for updates.
TXOK	**Transmitted Message Successfully** Indicates that a message has been transmitted successfully (error free and acknowledged by at least one other node), since this bit was last reset by the CPU (the CAN controller does not reset this bit!).
RXOK	**Received Message Successfully** This bit is set each time a message has been received successfully, since this bit was last reset by the CPU (the CAN controller does not reset this bit!). RXOK is also set when a message is received that is not accepted (i.e. stored).
EWRN	**Error Warning Status** Indicates that at least one of the error counters in the EML has reached the error warning limit of 96.
BOFF	**Busoff Status** Indicates when the CAN controller is in busoff state (see EML).

Note: Reading the upper half of the Control Register (status partition) will clear the Status Change Interrupt value in the Interrupt Register, if it is pending. Use byte accesses to the lower half to avoid this.

CAN Interrupt Handling

The on-chip CAN module has one interrupt output, which is connected (through a synchronization stage) to a standard interrupt node in the C164 in the same manner as all other interrupts of the standard on-chip peripherals. With this configuration, the user has all control options available for this interrupt, such as enabling/disabling, level and group priority, and interrupt or PEC service (see note below). The on-chip CAN module is connected to an XBUS interrupt control register.
As for all other interrupts, the node interrupt request flag is cleared automatically by hardware when this interrupt is serviced (either by standard interrupt or PEC service).

Note: As a rule, CAN interrupt requests can be serviced by a PEC channel. However, because PEC channels only can execute single predefined data transfers (there are no conditional PEC transfers), PEC service can only be used, if the request is known to be generated by one specific source, and that no other interrupt request will be generated in between. In practice this seems to be a rare case.

Since an interrupt request of the CAN module can be generated due to different conditions, the appropriate CAN interrupt status register must be read in the service routine to determine the cause of the interrupt request. The interrupt identifier INTID (a number) in the Port Control / Interrupt Register (PCIR) indicates the cause of an interrupt. When no interrupt is pending, the identifier will have the value 00_H.

If the value in INTID is not 00_H, then there is an interrupt pending. If bit IE in the control/ status register is set also the interrupt signal to the CPU is activated. The interrupt signal (to the interrupt node) remains active until INTID gets 00_H (i.e. all interrupt requests have been serviced) or until interrupt generation is disabled (CSR.IE = ’0’).

Note: The interrupt node is activated only upon a 0-->1 transition of the CAN interrupt signal. The CAN interrupt service routine should only be left after INTID has been verified to be 00_H*.*

The interrupt with the lowest number has the highest priority. If a higher priority interrupt (lower number) occurs before the current interrupt is processed, INTID is updated and the new interrupt overrides the last one.

INTID is also updated when the respective source request has been processed. This is indicated by clearing the INTPND flag in the respective object’s message control register (MCRn) or by reading the status partition of register CSR (in case of a status change interrupt). The updating of INTID is done by the CAN state machine and takes up to 6 CAN clock cycles (1 CAN clock cycle = 1 or 2 CPU clock cycles, determined by the prescaler bit CPS), depending on current state of the state machine.

Note: A worst case condition can occur when BRP = 00_H ***AND*** *the CAN controller is storing a just received message* ***AND*** *the CPU is executing consecutive accesses to the CAN module. In this rare case the maximum delay may be 26 CAN clock cycles.*
The impact of this delay can be minimized by clearing bit INTPND at an early stage

of interrupt processing, and (if required) restricting CPU accesses to the CAN module until the anticipated updating is complete.

PCIR

Port Control / Interrupt Register XReg (EF02$_H$) Reset value: XXXX$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
- reserved -					**IPC**			**INTID**							
-	-	-	-	-		rw		rh							

Bit	Function	
INTID	**Interrupt Identifier** This number indicates the cause of the interrupt (if pending).	
	00$_H$	**Interrupt Idle:** There is no interrupt request pending.
	01$_H$	**Status Change Interrupt:** The CAN controller has updated (not necessarily changed) the status in the Control Register. This can refer to a change of the error status of the CAN controller (EIE is set and BOFF or EWRN change) or to a CAN transfer incident (SIE must be set), like reception or transmission of a message (RXOK or TXOK is set) or the occurrence of a CAN bus error (LEC is updated). The CPU may clear RXOK, TXOK, and LEC, however, writing to the status partition of the Control Register can never generate or reset an interrupt. To update the INTID value the status partition of the Control Register must be read.
	02$_H$	**Message 15 Interrupt:** Bit INTPND in the Message Control Register of message object 15 (last message) has been set. The last message object has the highest interrupt priority of all message objects. [1]
	(02+N)	**Message N Interrupt:** Bit INTPND in the Message Control Register of message object ‘N’ has been set (N = 1...14). Note that a message interrupt code is only displayed, if there is no other interrupt request with a higher priority. [1] Example: message 1: INTID = 03$_H$, message 14: INTID = 10$_H$
IPC	**Interface Port Control** (reset value = 111$_B$, i.e. no port connection) The encoding of bitfield IPC is described in section “The CAN Application Interface“. *Note: Bitfield IPC can be written only while bit CCE is set.*	

[1] Bit INTPND of the corresponding message object has to be cleared to give messages with a lower priority the possibility to update INTID or to reset INTID to “00$_H$” (idle state).

Configuration of the Bit Timing

According to the CAN protocol specification, a bit time is subdivided into four segments: Sync segment, propagation time segment, phase buffer segment 1 and phase buffer segment 2.

Each segment is a multiple of the time quantum t_q, with $t_q = (BRP+1) * 2^{(1-CPS)} * t_{XCLK}$.

The Synchronization Segment (Sync Seg) is always 1 t_q long. The Propagation Time Segment and the Phase Buffer Segment 1 (combined to TSeg1) define the time before the sample point, while Phase Buffer Segment 2 (TSeg2) defines the time after the sample point. The length of these segments is programmable (except Sync-Seg) via the Bit Timing Register (BTR).

Note: For exact definition of these segments please refer to the CAN Protocol Specification.

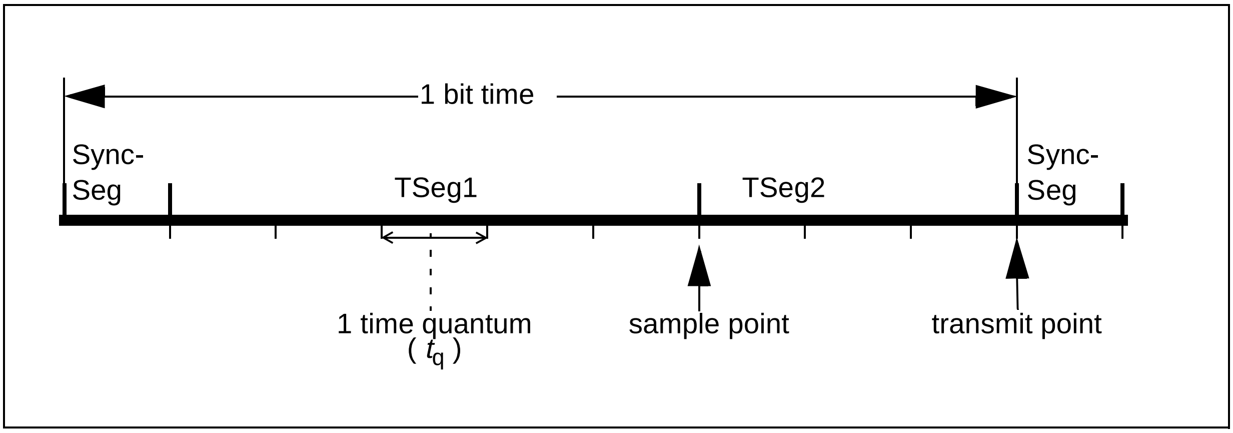


Figure 19-4 Bit Timing Definition

The bit time is determined by the XBUS clock period t_{XCLK}, the Baud Rate Prescaler, and the number of time quanta per bit:

$$\text{bit time} = t_{Sync\text{-}Seg} + t_{TSeg1} + t_{TSeg2}$$

$$t_{Sync\text{-}Seg} = 1 \bullet t_q$$

$$t_{TSeg1} = (TSEG1 + 1) \bullet t_q$$

$$t_{TSeg2} = (TSEG2 + 1) \bullet t_q$$

$$t_q = (BRP + 1) \bullet 2^{(1-CPS)} \bullet t_{XCLK}$$

Note: TSEG1, TSEG2, and BRP are the programmed numerical values from the respective fields of the Bit Timing Register.

BTR

Bit Timing Register **XReg (EF04$_H$)** **Reset value: UUUU$_H$**

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td colspan="3">TSEG2</td><td colspan="4">TSEG1</td><td colspan="2">SJW</td><td colspan="6">BRP</td></tr>
<tr><td>r</td><td colspan="3">rw</td><td colspan="4">rw</td><td colspan="2">rw</td><td colspan="6">rw</td></tr>
</table>

Bit	Function
BRP	**Baud Rate Prescaler** For generating the bit time quanta the CPU frequency f_{CPU} is divided by $2^{(1-CPS)}$ * (BRP+1). See also the prescaler control bit CPS in register CSR.
SJW	**(Re)Synchronization Jump Width** Adjust the bit time by maximum (SJW+1) time quanta for resynchronization.
TSEG1	**Time Segment before sample point** There are (TSEG1+1) time quanta before the sample point. Valid values for TSEG1 are “2...15”.
TSEG2	**Time Segment after sample point** There are (TSEG2+1) time quanta after the sample point. Valid values for TSEG2 are “1...7”.

Note: This register can only be written, if the config. change enable bit (CCE) is set.

Hard Synchronization and Resynchronization

To compensate phase shifts between clock oscillators of different CAN controllers, any CAN controller has to synchronize on any edge from recessive to dominant bus level if the edge lies between a Sample Point and the next Synchronization Segment, and on any other edge if it itself does not send a dominant level. If the Hard Synchronization is enabled (at the Start of Frame), the bit time is restarted at the Synchronization Segment, otherwise the Resynchronization Jump Width (SJW) defines the maximum number of time quanta by which a bit time may be shortened or lengthened during one Resynchronization. The current bit time is adjusted by

$$t_{SJW} = (SJW + 1) \bullet t_q$$

Note: SJW is the programmed numerical value from the respective field of the Bit Timing Register.

Calculation of the Bit Time

The programming of the bit time according to the CAN Specification depends on the desired baudrate, the XCLK frequency, and on the external physical delay times of the bus driver, of the bus line and of the input comparator. These delay times are summarized in the Propagation Time Segment t_{Prop}, where

t_{Prop} is two times the maximum of the sum of physical bus delay, the input comparator delay, and the output driver delay rounded up to the nearest multiple of t_{q}.

To fulfill the requirements of the CAN specification, the following conditions must be met:

$$t_{\mathrm{TSeg2}} \geq 2 \bullet t_{\mathrm{q}} = \textit{Information Processing Time}$$
$$t_{\mathrm{TSeg2}} \geq t_{\mathrm{SJW}}$$
$$t_{\mathrm{TSeg1}} \geq 3 \bullet t_{\mathrm{q}}$$
$$t_{\mathrm{TSeg1}} \geq t_{\mathrm{SJW}} + t_{\mathrm{Prop}}$$

Note: In order to achieve correct operation according to the CAN protocol the total bit time should be at least 8 t_q, i.e. TSEG1 + TSEG2 $\geq$ 5.
So, to operate with a baudrate of 1 MBit/sec, the XCLK frequency has to be at least 8/16 MHz (depending on the prescaler control bit CPS in register CSR).

The maximum tolerance df for XCLK depends on the Phase Buffer Segment 1 (PB1), the Phase Buffer Segment 2 (PB2), and the Resynchronization Jump Width (SJW):

$$df \leq \frac{\min(\mathrm{PB1}, \mathrm{PB2})}{2 \times (13 \times \text{bit time} - \mathrm{PB2})}$$

AND

$$df \leq \frac{tSJW}{20 \times \text{bit time}}$$

The examples below show how the bit timing is to be calculated under specific circumstances.

Bit Timing Example for High Baudrate

This example makes the following assumptions:

- XCLK frequency = 20 MHz
- BRP = 00, CPS = 0
- Baudrate = 1 Mbit/sec

t_{q}	100 ns	$= 2 \bullet t_{\mathrm{XCLK}}$
bus driver delay	50 ns	
receiver circuit delay	30 ns	
bus line (40 m) delay	220 ns	
t_{Prop}	600 ns	$= 6 \bullet t_{\mathrm{q}}$
t_{SJW}	100 ns	$= 1 \bullet t_{\mathrm{q}}$
t_{TSeg1}	700 ns	$= t_{\mathrm{Prop}} + t_{\mathrm{SJW}}$
t_{TSeg2}	200 ns	= *Information Processing Time*
t_{Sync}	100 ns	$= 1 \bullet t_{\mathrm{q}}$
t_{Bit}	1000 ns	$= t_{\mathrm{Sync}} + t_{\mathrm{TSeg1}} + t_{\mathrm{TSeg2}}$
tolerance for f_{XCLK}	0.39%	$= \dfrac{min(PB1, PB2)}{2 \times (13 \times \text{bit time} - \text{PB2})}$
		$= \dfrac{0{,}1\mu s}{2 \times (13 \times 1\mu s - 0{,}2\mu s)}$

Bit Timing Example for Low Baudrate

This example makes the following assumptions:

- XCLK frequency = 4 MHz
- BRP = 01, CPS = 0
- Baudrate = 100 kbit/sec

t_{q}	1 µs	$= 4 \bullet t_{\mathrm{XCLK}}$
bus driver delay	200 ns	
receiver circuit delay	80 ns	
bus line (40 m) delay	220 ns	
t_{Prop}	1 µs	$= 1 \bullet t_{\mathrm{q}}$
t_{SJW}	4 µs	$= 4 \bullet t_{\mathrm{q}}$
t_{TSeg1}	5 µs	$= t_{\mathrm{Prop}} + t_{\mathrm{SJW}}$
t_{TSeg2}	4 µs	= *Information Processing Time* $+ 2 \bullet t_{\mathrm{q}}$
t_{Sync}	1 µs	$= 1 \bullet t_{\mathrm{q}}$
t_{Bit}	10 µs	$= t_{\mathrm{Sync}} + t_{\mathrm{TSeg1}} + t_{\mathrm{TSeg2}}$
tolerance for f_{XCLK}	1.58%	$= \dfrac{min(PB1, PB2)}{2 \times (13 \times \text{bit time} - \text{PB2})}$
		$= \dfrac{4\mu s}{2 \times (13 \times 10\mu s - 4\mu s)}$

Mask Registers

Messages can use standard or extended identifiers. Incoming frames are masked with their appropriate global masks. Bit IDE of the incoming message determines, if the standard 11-bit mask in Global Mask Short (GMS) is to be used, or the 29-bit extended mask in Global Mask Long (UGML&LGML). Bits holding a “0” mean “don’t care”, i.e. do not compare the message’s identifier in the respective bit position.

The last message object (15) has an additional individually programmable acceptance mask (Mask of Last Message, UMLM&LMLM) for the complete arbitration field. This allows classes of messages to be received in this object by masking some bits of the identifier.

Note: The Mask of Last Message is ANDed with the Global Mask that corresponds to the incoming message.

GMS
Global Mask Short **XReg (EF06$_H$)** **Reset value: UFUU$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID20...18			**1**	**1**	**1**	**1**	**1**	**ID28...21**							
rw			r	r	r	r	r	rw							

Bit	Function
ID28...18	**Identifier (11-bit)** Mask to filter incoming messages with standard identifier.

UGML
Upper Global Mask Long **XReg ($EF08_H$)** **Reset value: $UUUU_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID20...13								**ID28...21**							
rw								rw							

LGML
Lower Global Mask Long **XReg ($EF0A_H$)** **Reset value: $UUUU_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID4...0					**0**	**0**	**0**	**ID12...5**							
rw					r	r	r	rw							

Bit	Function
ID28...0	**Identifier (29-bit)** Mask to filter incoming messages with extended identifier.

UMLM
Upper Mask of Last Message **XReg ($EF0C_H$)** **Reset value: $UUUU_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID20...18			**ID17...13**					**ID28...21**							
rw			rw					rw							

LMLM
Lower Mask of Last Message **XReg ($EF0E_H$)** **Reset value: $UUUU_H$**

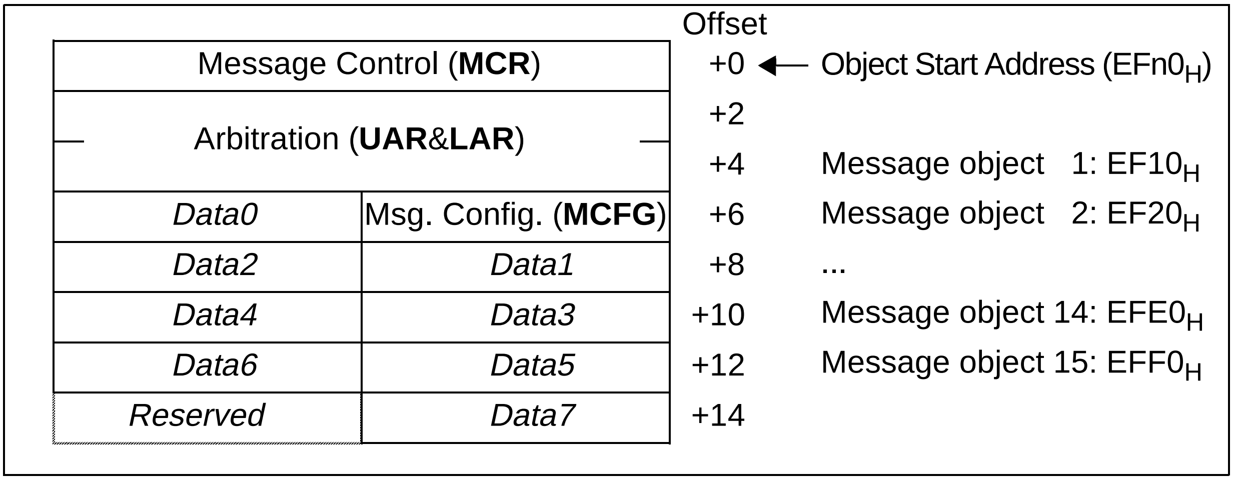
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID4...0					**0**	**0**	**0**	**ID12...5**							
rw					r	r	r	rw							

Bit	Function
ID28...0	**Identifier (29-bit)** Mask to filter the last incoming message (Nr. 15) with standard or extended identifier (as configured).

19.3 The Message Object

The message object is the primary means of communication between CPU and CAN controller. Each of the 15 message objects uses 15 consecutive bytes (see map below) and starts at an address that is a multiple of 16.

Note: All message objects must be initialized by the CPU, even those which are not going to be used, before clearing the INIT bit.



		Offset
Message Control (**MCR**)		+0
Arbitration (**UAR&LAR**)		+2
		+4
Data0	Msg. Config. (**MCFG**)	+6
Data2	*Data1*	+8
Data4	*Data3*	+10
Data6	*Data5*	+12
Reserved	*Data7*	+14

+0 ← Object Start Address (EFn0$_H$)

Message object 1: EF10$_H$
Message object 2: EF20$_H$
...
Message object 14: EFE0$_H$
Message object 15: EFF0$_H$

Figure 19-5 Message Object Address Map

The general properties of a message object are defined via the Message Control Register (MCR). There is a dedicated register MCRn for each message object n.

Each element of the Message Control Register is made of two complementary bits.This special mechanism allows the selective setting or resetting of specific elements (leaving others unchanged) without requiring read-modify-write cycles. None of these elements will be affected by reset.
The table below shows how to use and interpret these 2-bit fields.

Table 19-1 MCR Bitfield Encoding

Value	Function on Write	Meaning on Read
0 0	-reserved-	-reserved-
0 1	Reset element	Element is reset
1 0	Set element	Element is set
1 1	Leave element unchanged	-reserved-

MCRn
Message Control Register **XReg (EFn0$_H$)** **Reset value: UUUU$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RMTPND		**TXRQ**		**MSGLST** **CPUUPD**		**NEWDAT**		**MSGVAL**		**TXIE**		**RXIE**		**INTPND**	
rw		rw		rw		rw		rw		rw		rw		rw	

Bit	Function
INTPND	**Interrupt Pending** Indicates, if this message object has generated an interrupt request (see TXIE and RXIE), since this bit was last reset by the CPU, or not.
RXIE	**Receive Interrupt Enable** Defines, if bit INTPND is set after successful reception of a frame.
TXIE	**Transmit Interrupt Enable** Defines, if bit INTPND is set after successful transmission of a frame. [1]
MSGVAL	**Message Valid** Indicates, if the corresponding message object is valid or not. The CAN controller only operates on valid objects. Message objects can be tagged invalid, while they are changed, or if they are not used at all.
NEWDAT	**New Data** Indicates, if new data has been written into the data portion of this message object by CPU (transmit-objects) or CAN controller (receive-objects) since this bit was last reset, or not. [2]
MSGLST	**Message Lost** (This bit applies to receive-objects only!) Indicates that the CAN controller has stored a new message into this object, while NEWDAT was still set, i.e. the previously stored message is lost.
CPUUPD	**CPU Update** (This bit applies to transmit-objects only!) Indicates that the corresponding message object may not be transmitted now. The CPU sets this bit in order to inhibit the transmission of a message that is currently updated, or to control the automatic response to remote requests.
TXRQ	**Transmit Request** Indicates that the transmission of this message object is requested by the CPU or via a remote frame and is not yet done. TXRQ can be disabled by CPUUPD. [1] [3]

Bit	Function
RMTPND	**Remote Pending** (Used for transmit-objects) Indicates that the transmission of this message object has been requested by a remote node, but the data has not yet been transmitted. When RMTPND is set, the CAN controller also sets TXRQ. RMTPND and TXRQ are cleared, when the message object has been successfully transmitted.

[1] In message object 15 (last message) these bits are hardwired to “0” (inactive) in order to prevent transmission of message 15.

[2] When the CAN controller writes new data into the message object, unused message bytes will be overwritten by non specified values. Usually the CPU will clear this bit before working on the data, and verify that the bit is still cleared once it has finished working to ensure that it has worked on a consistent set of data and not part of an old message and part of the new message.
For transmit-objects the CPU will set this bit along with clearing bit CPUUPD. This will ensure that, if the message is actually being transmitted during the time the message was being updated by the CPU, the CAN controller will not reset bit TXRQ. In this way bit TXRQ is only reset once the actual data has been transferred.

[3] When the CPU requests the transmission of a receive-object, a remote frame will be sent instead of a data frame to request a remote node to send the corresponding data frame. This bit will be cleared by the CAN controller along with bit RMTPND when the message has been successfully transmitted, if bit NEWDAT has not been set.
If there are several valid message objects with pending transmission request, the message with the lowest message number is transmitted first. This arbitration is done when several objects are requested for transmission by the CPU, or when operation is resumed after an error frame or after arbitration has been lost.

Arbitration Registers

The Arbitration Registers (UARn&LARn) are used for acceptance filtering of incoming messages and to define the identifier of outgoing messages. A received message with a matching identifier is accepted as a data frame (matching object has DIR=’0’) or as a remote frame (matching object has DIR=’1’). For matching, the corresponding Global Mask has to be considered (in case of message object 15 also the Mask of Last Message). Extended frames (using Global Mask Long) can be stored only in message objects with XTD=’1’, standard frames (using Global Mask Short) only in message objects with XTD=’0’.

Message objects should have unique identifiers, i.e. if some bits are masked out by the Global Mask Registers (i.e. “don’t care”), then the identifiers of the valid message objects should differ in the remaining bits which are used for acceptance filtering.

If a received message (data frame or remote frame) matches with more than one valid message object, it is associated with the object with the lowest message number. I.e. a received data frame is stored in the “lowest” object, or the “lowest” object is sent in response to a remote frame. The Global Mask is used for matching here.

After a transmission (data frame or remote frame) the transmit request flag of the matching object with the lowest message number is cleared. The Global Mask is not used in this case.

When the CAN controller accepts a data frame, the complete message is stored into the corresponding message object, including the identifier (also masked bits, standard identifiers have bits ID17-0 filled with ’0’), the data length code (DLC), and the data bytes (valid bytes indicated by DLC). This is implemented to keep the data bytes connected with the identifier, even if arbitration mask registers are used.

When the CAN controller accepts a remote frame, the corresponding transmit message object (1...14) remains unchanged, except for bits TXRQ and RMTPND, which are set, of course. In the last message object 15 (which cannot start a transmission) the identifier bits corresponding to the “don’t care” bits of the Last Message Mask are copied from the received frame. Bits corresponding to the “don’t care” bits of the corresponding global mask are not copied (i.e. bits masked out by the global **and** the last message mask cannot be retrieved from object 15).

UARn
Upper Arbitration Register **XReg (EFn2$_H$)** **Reset value: UUUU$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID20...18			**ID17...13**					**ID28...21**							
rw								rw							

LARn
Lower Arbitration Register **XReg (EFn4$_H$)** **Reset value: UUUU$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID4...0					**0**	**0**	**0**	**ID12...5**							
rw					r	r	r	rw							

Bit	Function
ID28...0	**Identifier (29-bit)** Identifier of a standard message (ID28...18) or an extended message (ID28...0). For standard identifiers bits ID17...0 are “don’t care”.

Message Configuration

The Message Configuration Register (low byte of MCFGn) holds a description of the message within this object.

Note: There is no “don’t care” option for bits XTD and DIR. So incoming frames can only match with corresponding message objects, either standard (XTD=0) or extended (XTD=1). Data frames only match with receive-objects, remote frames only match with transmit-objects.
When the CAN controller stores a data frame, it will write all the eight data bytes into a message object. If the data length code was less than 8, the remaining bytes of the message object will be overwritten by non specified values.

MCFGn
Message Configuration Reg. XReg (EFn6$_H$) Reset value: - - UU$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data Byte 0								DLC				DIR	XTD	0	0
rw								rw				rw	rw	r	r

Bit	Function
XTD	**Extended Identifier** 0: **Standard** This message object uses a standard 11-bit identifier. 1: **Extended** This message object uses an extended 29-bit identifier.
DIR	**Message Direction** 0: **Receive object.** On TXRQ, a remote frame with the identifier of this message object is transmitted. On reception of a data frame with matching identifier, that message is stored in this message object. 1: **Transmit object.** On TXRQ, the respective message object is transmitted. On reception of a remote frame with matching identifier, the TXRQ and RMTPND bits of this message object are set.
DLC	**Data Length Code** Defines the number of valid data bytes within the data area. Valid values for the data length are 0...8.

Note: The first data byte occupies the upper half of the message configuration register.

Data Area

The data area occupies 8 successive byte positions after the Message Configuration Register, i.e. the data area of message object **n** covers locations 00’EFn7$_H$ through 00’EFnE$_H$.
Location 00’EFnF$_H$ is reserved.

Message data for message object 15 (last message) will be written into a two-message-alternating buffer to avoid the loss of a message, if a second message has been received, before the CPU has read the first one.

Handling of Message Objects

The following diagrams summarize the actions that have to be taken in order to transmit and receive messages over the CAN bus. The actions taken by the CAN controller are described as well as the actions that have to be taken by the CPU (i.e. the servicing program).

The diagrams show...

- CAN controller handling of transmit objects
- CAN controller handling of receive objects
- CPU handling of transmit objects
- CPU handling of receive objects
- CPU handling of last message object
- Handling of the last message’s alternating buffer

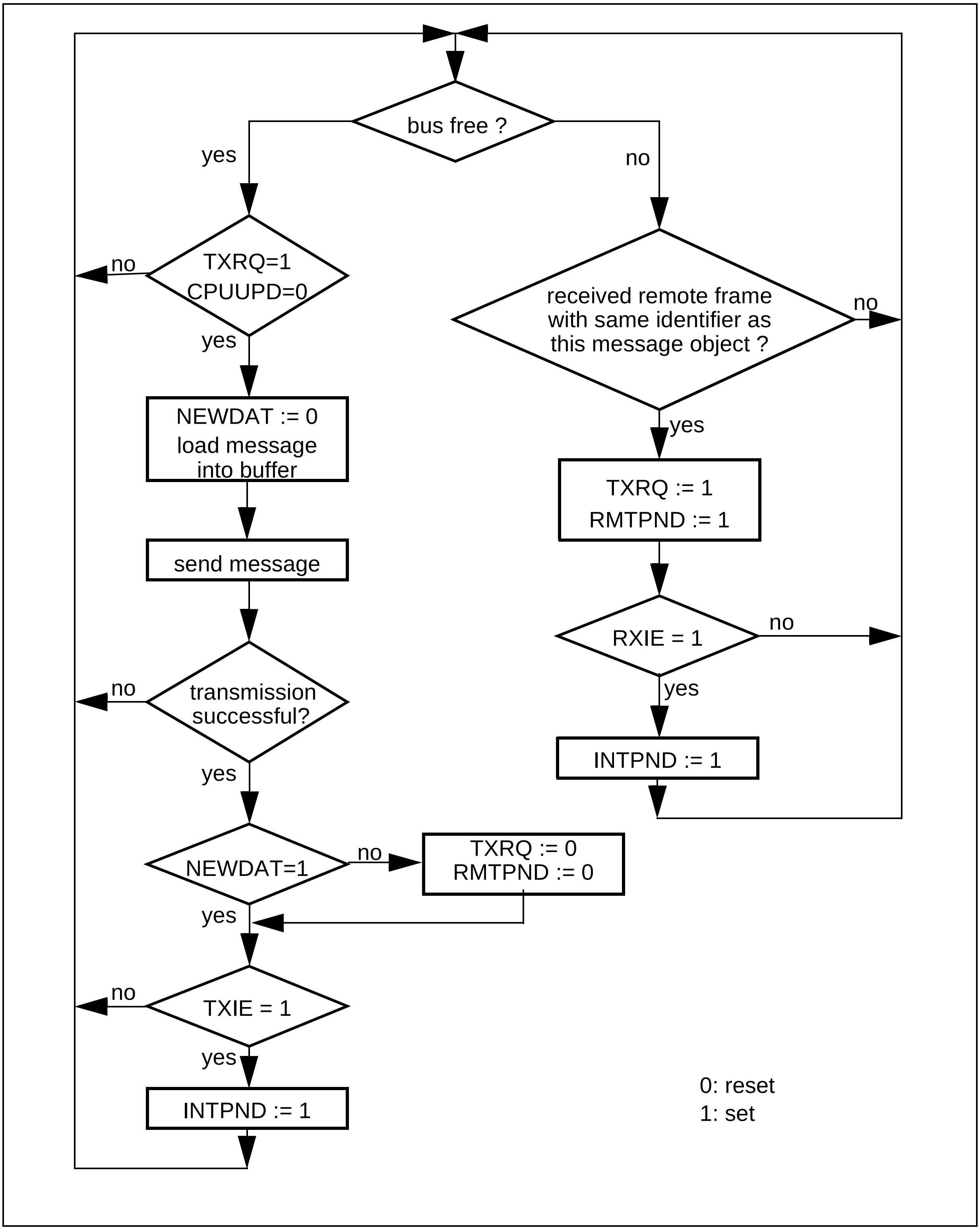


Figure 19-6 CAN Controller Handling of Transmit Objects (DIR = ’1’)

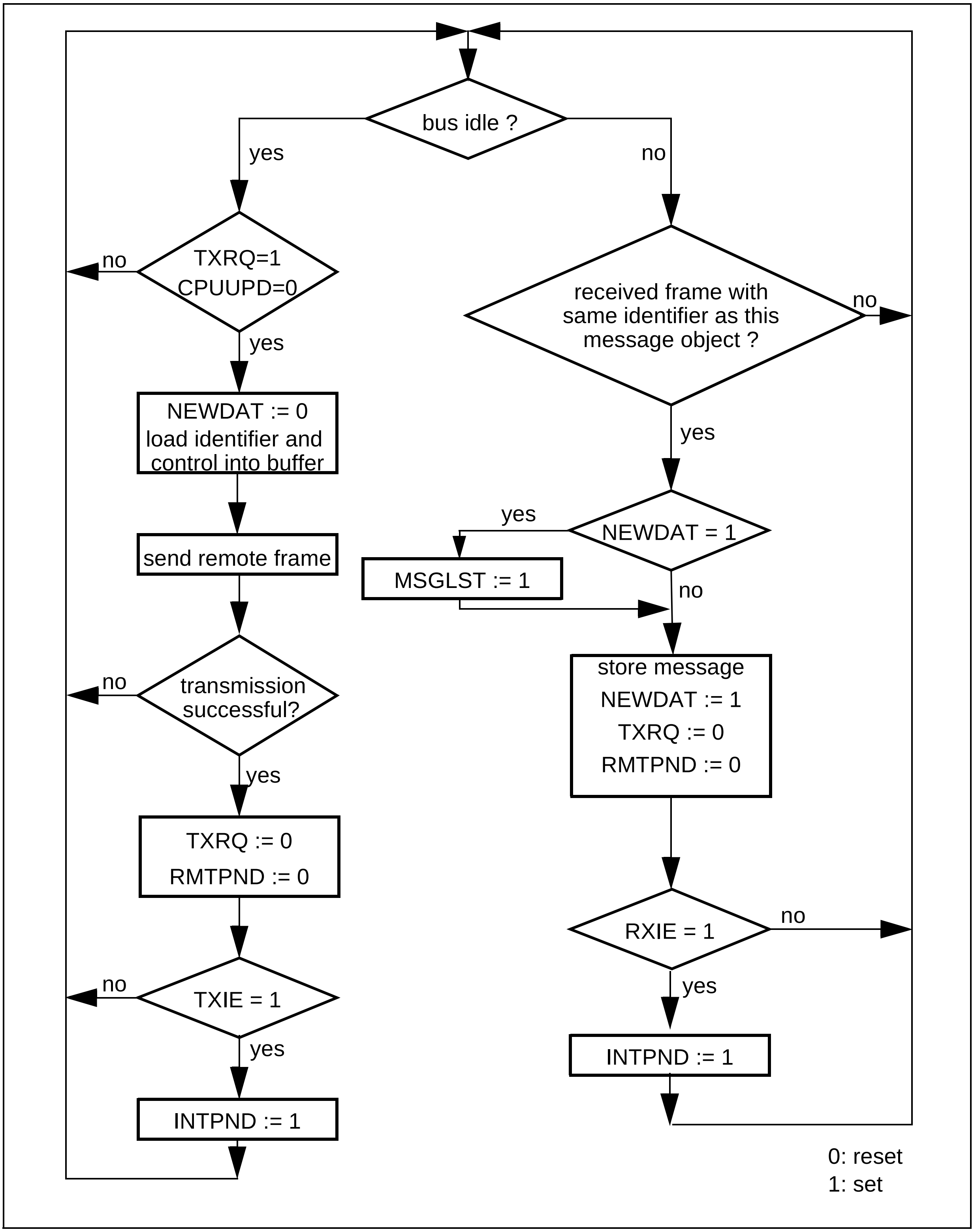


Figure 19-7 CAN Controller Handling of Receive Objects (DIR = ’0’)

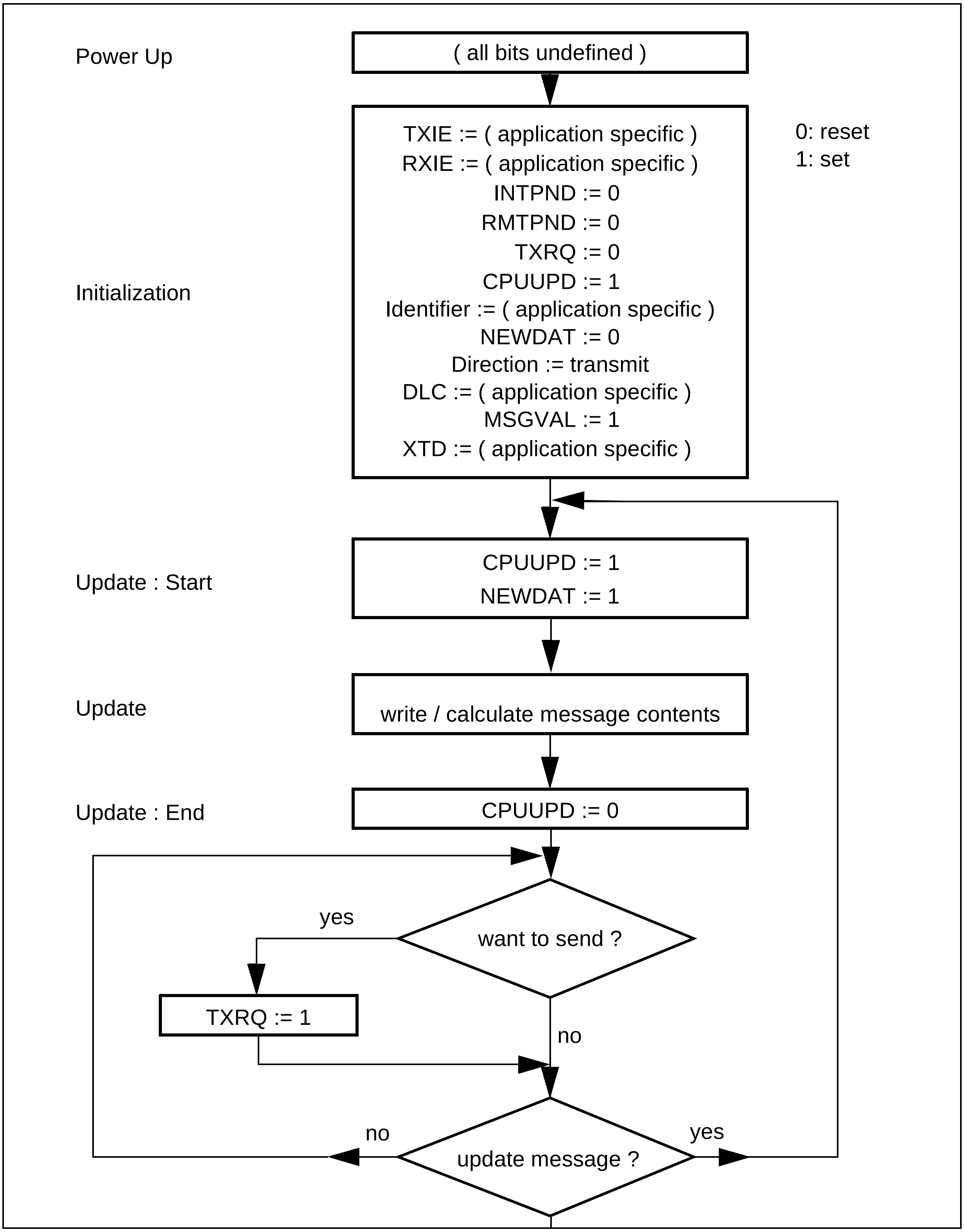


Figure 19-8 CPU Handling of Transmit Objects (DIR = '1')

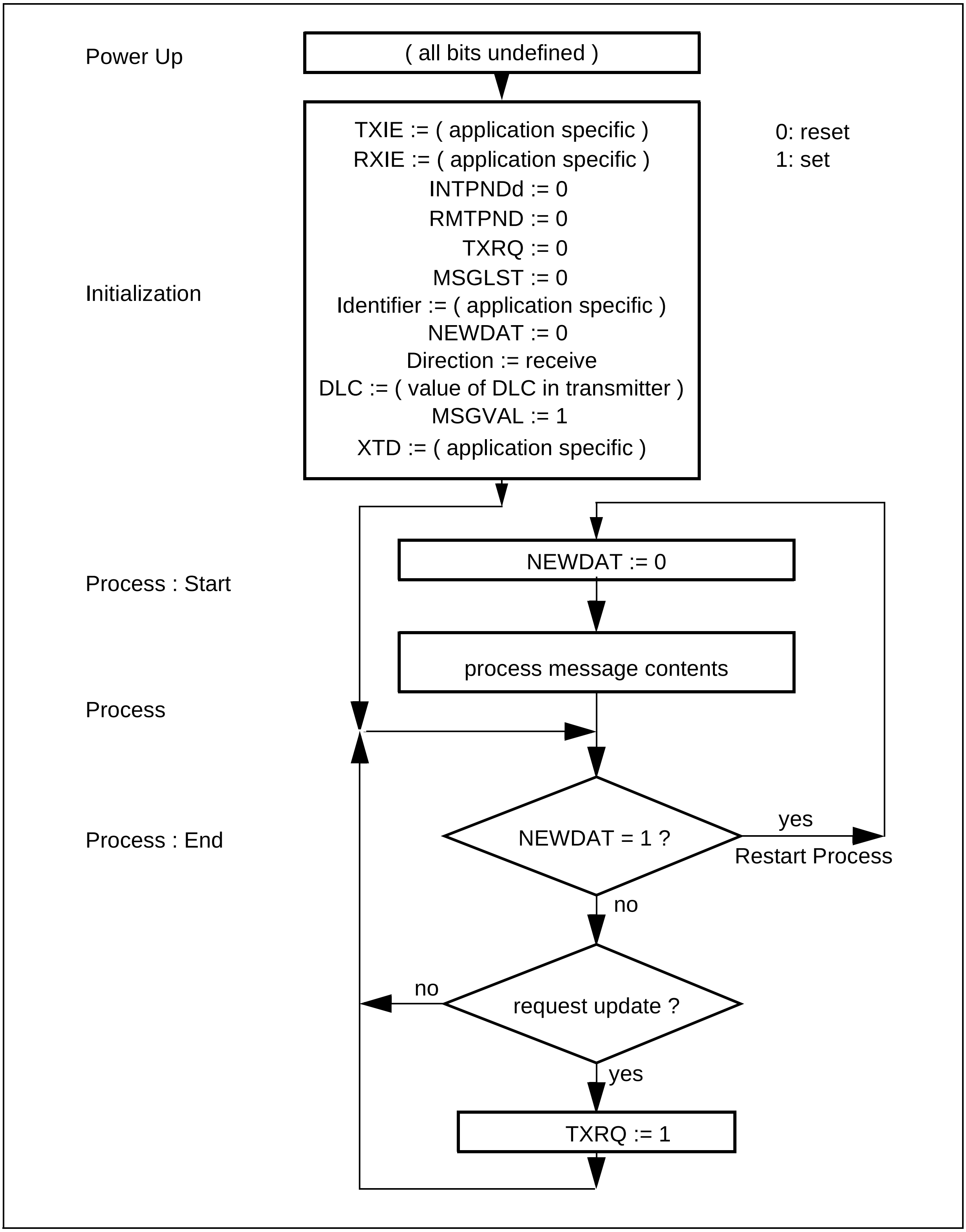


Figure 19-9 CPU Handling of Receive Objects (DIR = ’0’)

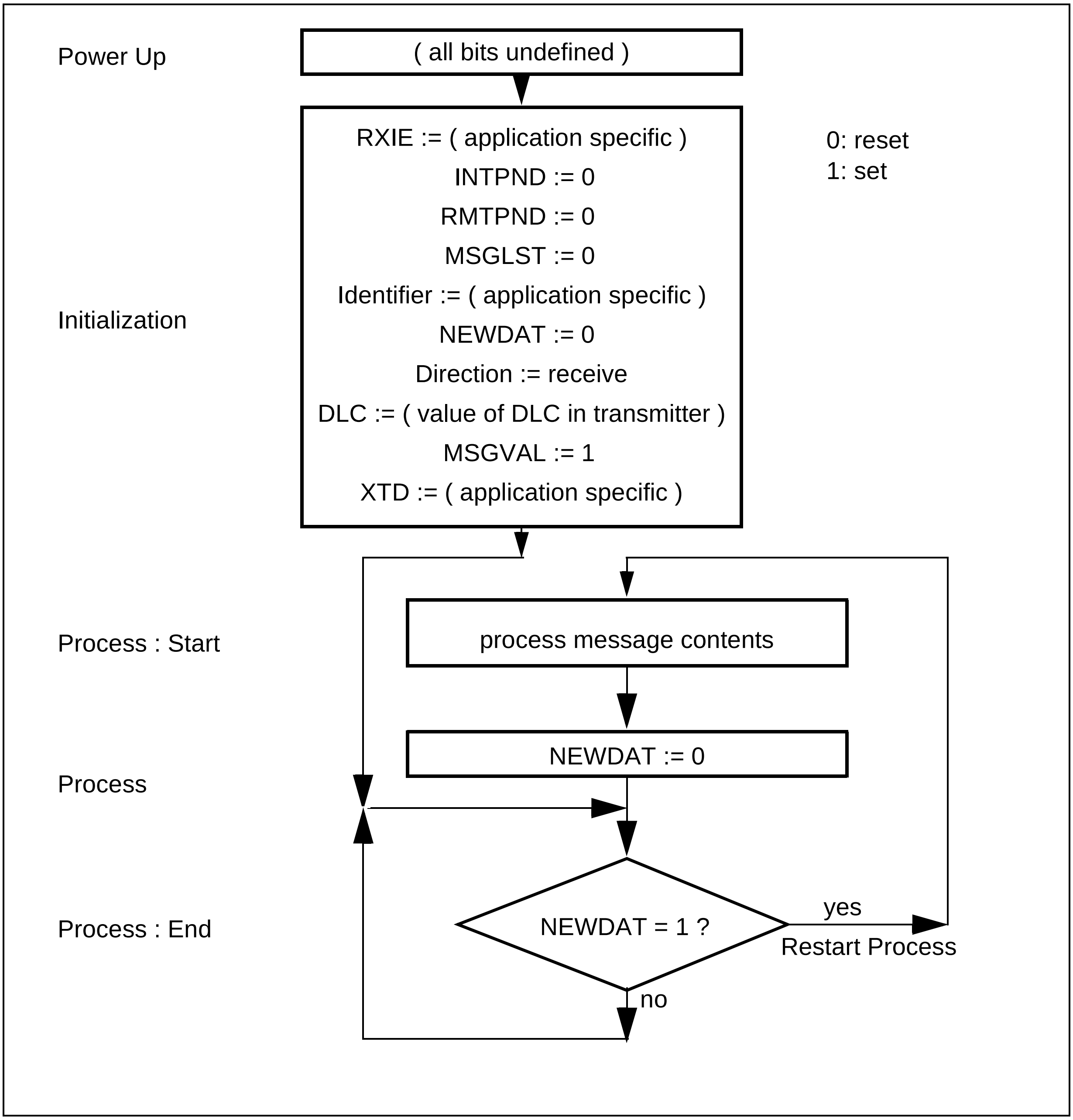
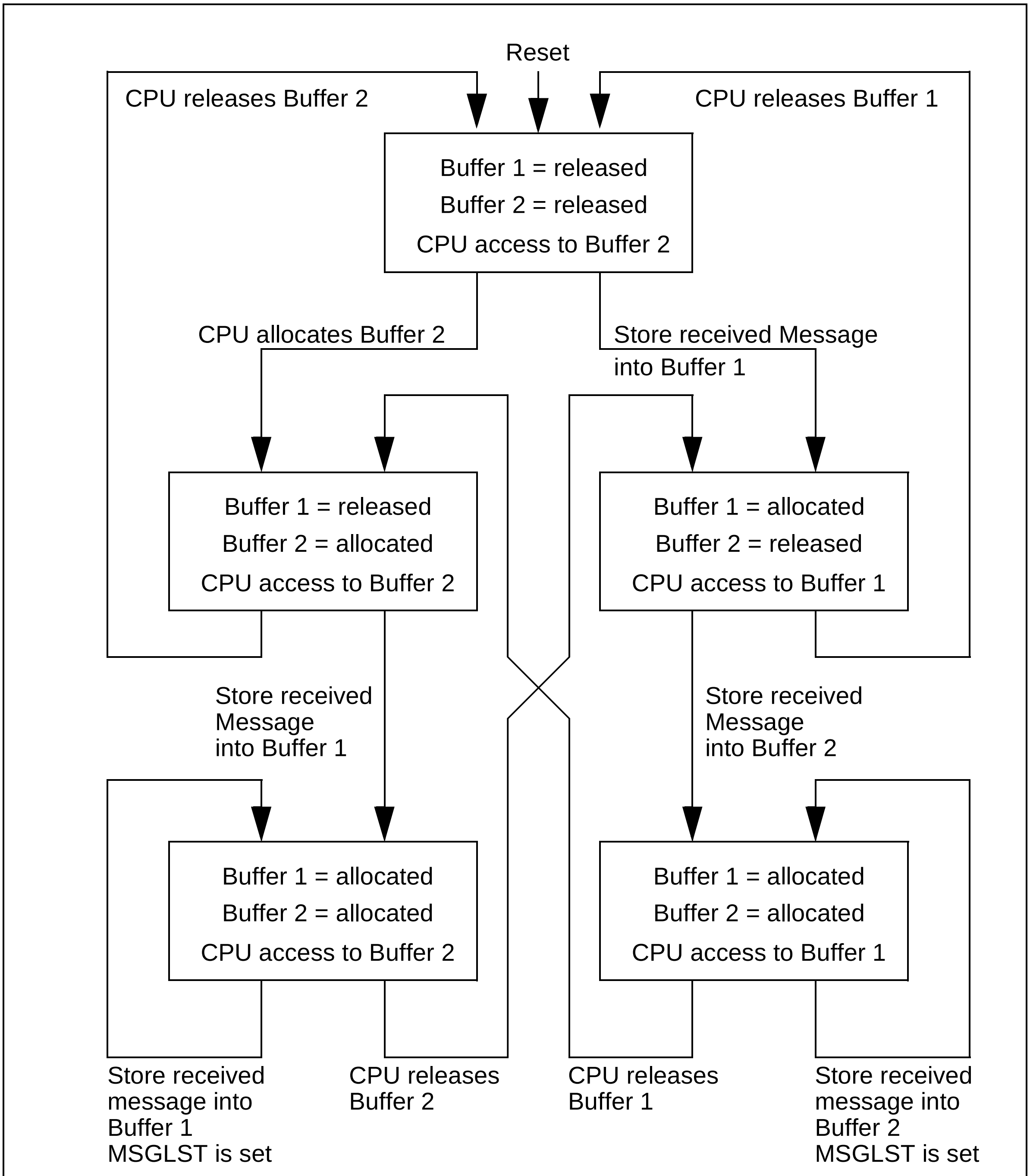


Figure 19-10 CPU Handling of the Last Message Object



Allocated	: NEWDAT = 1	OR	RMTPND = 1
Released	: NEWDAT = 0	AND	RMTPND = 0

Figure 19-11 Handling of the Last Message Object's Alternating Buffer

19.4 Controlling the CAN Module

The CAN module is controlled by the C164 via hardware signals (e.g. reset) and via register accesses executed by software.

Accessing the On-chip CAN Module

The CAN module is implemented as an X-Peripheral and is therefore accessed like an external memory or peripheral. That means that the registers of the CAN module can be read and written using 16-bit or 8-bit direct or indirect MEM addressing modes. Also bit handling is not supported via the XBUS. Since the XBUS, to which the CAN module is connected, also represents the external bus, CAN accesses follow the same rules and procedures as accesses to the external bus. CAN accesses cannot be executed in parallel to external instruction fetches or data read/writes, but are arbitrated and inserted into the external bus access stream.

Accesses to the CAN module use demultiplexed addresses, a 16-bit data bus (byte accesses possible), two waitstates and no tristate waitstate.

The CAN address area starts at 00’EF00$_H$ and covers 256 Bytes. This area is decoded internally, so none of the programmable address windows must be sacrificed in order to access the on-chip CAN module.

The advantage of locating the CAN address area in segment 0 is that the CAN module is accessible via data page 3, which is the 'system' data page, accessed usually through the 'system' data page pointer DPP3. In this way, the internal addresses, such like SFRs, internal RAM, and the CAN registers, are all located within the same data page and form a contiguous address space.

Power Down Mode

If the C164 enters Power Down mode, the XCLK signal will be turned off which will stop the operation of the CAN module. Any message transfer is interrupted. In order to ensure that the CAN controller is not stopped while sending a dominant level (’0’) on the CAN bus, the CPU should set bit INIT in the Control Register prior to entering Power Down mode. The CPU can check if a transmission is in progress by reading bits TXRQ and NEWDAT in the message objects and bit TXOK in the Control Register. After returning from Power Down mode via hardware reset, the CAN module has to be reconfigured.

Disabling the CAN Module

When the CAN module is disabled by setting bit CANDIS in register SYSCON3 (peripheral management) no register accesses are possible. Also the module's logic blocks are stopped and no CAN bus transfers are possible. After re-enabling the CAN module (CANDIS='0') it must be reconfigured (as after returning from Power Down mode).

Note: Incoming message frames can still be recognized (not received) in this case by monitoring the receive line CAN_RXD. For this purpose the receive line CAN_RXD can be connected to a fast external interrupt via register EXISEL.

CAN Module Reset

The on-chip CAN module is connected to the XBUS Reset signal. This signal is activated, when the C164's reset input is activated, when a software reset is executed, and in case of a watchdog reset. Activating the CAN module's reset line triggers a hardware reset.

This hardware reset...

- disconnects the CAN_TXD output from the port logic
- clears the error counters
- resets the busoff state
- switches the Control Register's low byte to 01_H
- leaves the Control Register's high byte and the Interrupt Register undefined
- does not change the other registers including the message objects (notified as UUUU)

*Note: The first hardware reset after power-on leaves the un**changed** registers in an un**defined** state, of course.*
The value 01_H in the Control Register's low byte prepares for the module initialization.

CAN Module Activation

After a reset the CAN module is disabled. Before it can be used to receive or transmit messages the application software must activate the CAN module.

Three actions are required for this purpose:

- **General Module Enable** globally activates the CAN module. This is done by setting bit XPEN in register SYSCON.
- **Pin Assignment** selects a pair of port pins that connect the CAN module to the external transceiver. This is done via bitfield IPC in register PCIR.
- **Module Initialization** determines the functionality of the CAN module (baudrate, active objects, etc.). This is the major part of the activation and is described in the following.

Module Initialization

The module initialization is enabled by setting bit INIT in the control register CSR. This can be done by the CPU via software, or automatically by the CAN controller on a hardware reset, or if the EML switches to busoff state.

While INIT is set...

- all message transfer from and to the CAN bus is stopped
- the CAN transmit line CAN_TXD is “1” (recessive)
- the control bits NEWDAT and RMTPND of the last message object are reset
- the counters of the EML are left unchanged.

Setting bit CCE in addition, permits changing the configuration in the Bit Timing Register.

To initialize the CAN Controller, the following actions are required:

- configure the Bit Timing Register (CCE required)
- set the Global Mask Registers
- initialize each message object.

If a message object is not needed, it is sufficient to clear its message valid bit (MSGVAL), i.e. to define it as not valid. Otherwise, the whole message object has to be initialized.

After the initialization sequence has been completed, the CPU clears bit INIT.

Now the BSP synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (i.e. Bus Idle) before it can take part in bus activities and start message transfers.

The initialization of the message objects is independent of the state of bit INIT and can be done on the fly. The message objects should all be configured to particular identifiers or set to “not valid” before the BSP starts the message transfer, however.

To change the configuration of a message object during normal operation, the CPU first clears bit MSGVAL, which defines it as not valid. When the configuration is completed, MSGVAL is set again.

Busoff Recovery Sequence

If the device goes *busoff*, it will set bit BOFF and also set bit INIT of its own accord, stopping all bus activities. To have the CAN module take part in the CAN bus activities again, the bus-off recovery sequence must be started by clearing the bit INIT (via software). Once INIT has been cleared, the module will then wait for 129 occurrences of *Bus Idle* before resuming normal operation.

At the end of the *busoff* recovery sequence the Error Management Counters will be reset. This will automatically clear bits BOFF and EWRN.

During the waiting time after the resetting of INIT each time a sequence of 11 recessive bits has been monitored, a **Bit0Error** code is written to the Control Register, enabling the CPU to check up whether the CAN bus is stuck at dominant or continously disturbed and to monitor the proceeding of the busoff recovery sequence.

Note: An interrupt can be generated when entering the busoff state if bits IE and EIE are set. The corresponding interrupt code in bitfield INTID is 01$_H$.
The busoff recovery sequence cannot be shortened by setting or resetting INIT.

19.5 Configuration Examples for Message Objects

The two examples below represent standard applications for using CAN messages. Both examples assume that identifier and direction are already set up correctly.

The respective contents of the Message Control Register (MCR) are shown.

Configuration Example of a Transmission Object

This object shall be configured for transmission. It shall be transmitted automatically in response to remote frames, but no receive interrupts shall be generated for this object.

MCR (Data bytes are not written completely --> CPUUPD = ’1’)

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
0 1	**0 1**	**1 0**	**0 1**	**1 0**	**0 1**	**0 1**	**0 1**
RMTPND	TXRQ	CPUUPD	NEWDAT	MSGVAL	TXIE	RXIE	INTPND

MCR (Remote frame was received in the meantime --> RMTPND = ’1’, TXRQ = ’1’)

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
1 0	**1 0**	**1 0**	**0 1**	**1 0**	**0 1**	**0 1**	**0 1**
RMTPND	TXRQ	CPUUPD	NEWDAT	MSGVAL	TXIE	RXIE	INTPND

After updating the message the CPU should clear CPUUPD and set NEWDAT. The previously received remote request will then be answered.

If the CPU wants to transmit the message actively it should also set TXRQ (which should otherwise be left alone).

Configuration Example of a Reception Object

This object shall be configured for reception. A receive interrupt shall be generated each time new data comes in. From time to time the CPU sends a remote request to trigger the sending of this data from a remote node.

MCR (Message object is idle, i.e. waiting for a frame to be received)

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
0 1	**0 1**	**0 1**	**0 1**	**1 0**	**0 1**	**1 0**	**0 1**
RMTPND	TXRQ	MSGLST	NEWDAT	MSGVAL	TXIE	RXIE	INTPND

.

MCR (A data frame was received --> NEWDAT = ’1’, INTPND = ’1’)

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
0 1	**0 1**	**0 1**	**1 0**	**1 0**	**0 1**	**1 0**	**1 0**
RMTPND	TXRQ	MSGLST	NEWDAT	MSGVAL	TXIE	RXIE	INTPND

To process the message the CPU should clear INTPND and NEWDAT, process the data, and check that NEWDAT is still clear after that. If not, the processing should be repeated.

To send a remote frame to request the data, simply bit TXRQ needs to be set. This bit will be cleared by the CAN controller, once the remote frame has been sent or if the data is received before the CAN controller could transmit the remote frame.

19.6 The CAN Application Interface

The on-chip CAN module of the C164 is connected to the (external) physical layer (i.e. the CAN bus) via two signals:

Table 19-2 CAN Interface Signals

CAN Signal	Port Pin	Function
CAN_RXD	Controlled via C1PCIR.IPC	Receive data from the physical layer of the CAN bus.
CAN_TXD		Transmit data to the physical layer of the CAN bus.

A logic low level ('0') is interpreted as the dominant CAN bus level, a logic high level ('1') is interpreted as the recessive CAN bus level.

Connection to an External Transceiver

The CAN module of the C164 can be connected to an external CAN bus via a CAN transceiver.

Note: Basically it is also possible to connect several CAN modules directly (on-board) without using CAN transceivers.

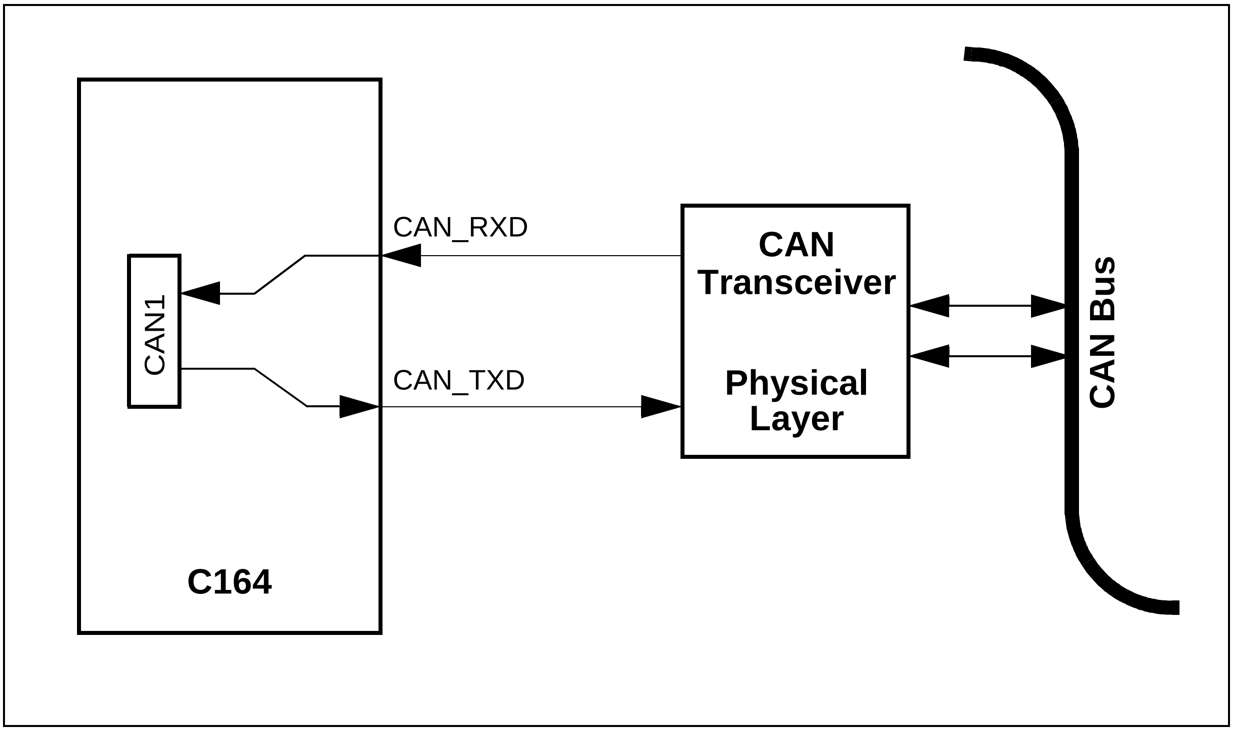


Figure 19-12 Connection to a Single CAN Bus

Port Control

The receive data line and the transmit data line of the CAN module are alternate port functions. Make sure that the respective port pin for the receive line is switched to input in order to enable proper reception. The respective port driver for the transmit will automatically be switched ON.

This provides a standard pin configuration without additional software control and also works in emulation mode where the port direction registers cannot be controlled.

The receive and transmit line of the CAN module may be assigned to several port pins of the C164 under software control. This assignment is selected via bitfield IPC (Interface Port Connection) in register PCIR.

Table 19-3 Assignment of CAN Interface Lines to Port Pins

IPC	CAN_RXD	CAN_TXD	Notes
000	P4.5	P4.6	Compatible assignments (CAN1). [1]
001	---	---	*Reserved*. Do not use this combination.
010	P8.0	P8.1	Port 4 available for segment address lines A21...A16 (4 MByte external address space).
011	P8.2	P8.3	Port 4 available for segment address lines A21...A16 (4 MByte external address space).
100	---	---	*Reserved*. Do not use this combination.
101	---	---	*Reserved*. Do not use this combination.
110	---	---	*Reserved*. Do not use this combination.
111	Idle (recessive)	Disconnected	No port assigned. Default after Reset.

[1] This assignment is compatible with previous derivatives where the assignment of CAN interface lines was fixed.

The location of the CAN interface lines can now be selected via software according to the requirements of an application:

Compatible Assignment (IPC=000_B) makes the C164 suitable for applications with a given hardware (board layout). The CAN interface lines are connected to the port pins to which they are hardwired in previous derivatives.

Full Address Assignment (IPC=010_B or 011_B) removes the CAN interface lines completely from Port 4. The maximum external address space of 4 MByte is available in this case.
The CAN interface lines are mapped to Port 8. Two pairs of Port 8 pins can be selected.

No Assignment (IPC=111_B) disconnects the CAN interface lines from the port logic. This avoids undesired currents through the interface pin drivers while the C164 is in a power saving state.
After reset the CAN interface lines are disconnected.

Note: Assigning CAN interface signals to a port pin overrides the other alternate function of the respective pin (segment address on Port 4, CAPCOM lines on Port 8).

20 System Reset

The internal system reset function provides initialization of the C164 into a defined default state and is invoked either by asserting a hardware reset signal on pin $\overline{\text{RSTIN}}$ (Hardware Reset Input), upon the execution of the SRST instruction (Software Reset) or by an overflow of the watchdog timer.

Whenever one of these conditions occurs, the microcontroller is reset into its predefined default state through an internal reset procedure. When a reset is initiated, pending internal hold states are cancelled and the current internal access cycle (if any) is completed. An external bus cycle is aborted, except for a watchdog reset (see description). After that the bus pin drivers and the IO pin drivers are switched off (tristate).

The internal reset procedure requires 516 CPU clock cycles in order to perform a complete reset sequence. This 516 cycle reset sequence is started upon a watchdog timer overflow, a SRST instruction or when the reset input signal $\overline{\text{RSTIN}}$ is latched low (hardware reset). The internal reset condition is active at least for the duration of the reset sequence and then until the $\overline{\text{RSTIN}}$ input is inactive and the PLL has locked (if the PLL is selected for the basic clock generation). When this internal reset condition is removed (reset sequence complete, $\overline{\text{RSTIN}}$ inactive, PLL locked) the reset configuration is latched from PORT0, $\overline{\text{RD}}$, and ALE (depending on the start mode). After that pins ALE, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ are driven to their inactive levels.

Note: Bit ADP which selects the Adapt mode is latched with the rising edge of $\overline{\text{RSTIN}}$.

After the internal reset condition is removed, the microcontroller will either start program execution from external or internal memory, or enter boot mode.

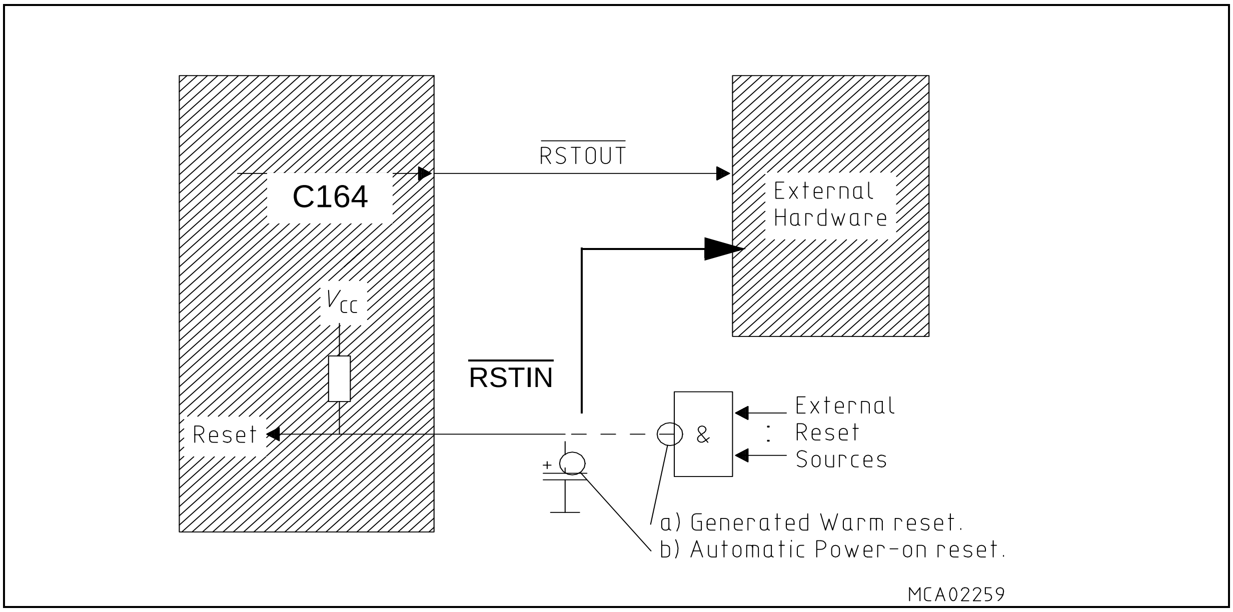


Figure 20-1 External Reset Circuitry

20.1 Reset Sources

Several sources (external or internal) can generate a reset for the C164. Software can identify the respective reset source via the reset source indication flags in register WDTCON. Generally any reset causes the same actions on the C164's modules. The differences are described in the following sections.

Hardware Reset

A hardware reset is triggered when the reset input signal $\overline{\text{RSTIN}}$ is latched low. To ensure the recognition of the $\overline{\text{RSTIN}}$ signal (latching), it must be held low for at least 100 ns plus 2 CPU clock cycles (input filter plus synchronization). Also shorter $\overline{\text{RSTIN}}$ pulses may trigger a hardware reset, if they coincide with the latch's sample point. The actual minimum duration for a reset pulse depends on the current CPU clock generation mode. The worstcase is generating the CPU clock via the SlowDown Divider using the maximum factor while the configured basic mode uses the prescaler ($f_{\text{CPU}} = f_{\text{OSC}}$ / 64 in this case).

After the reset sequence has been completed, the $\overline{\text{RSTIN}}$ input is sampled again. When the reset input signal is inactive at that time, the internal reset condition is terminated (indicated as short hardware reset, SHWR). When the reset input signal is still active at that time, the internal reset condition is prolonged until $\overline{\text{RSTIN}}$ gets inactive (indicated as long hardware reset, LHWR).

During a hardware reset the inputs for the reset configuration (PORT0, $\overline{\text{RD}}$, ALE) need some time to settle on the required levels, especially if the hardware reset aborts a read operation from an external peripheral. During this settling time the configuration may intermittently be wrong. For the duration of one internal reset sequence after a reset has been recognized the configuration latches are not transparent, i.e. the (new) configuration becomes valid earliest after the completion of one reset sequence. This usually covers the required settling time.

When the basic clock is generated by the PLL the internal reset condition is automatically extended until the on-chip PLL has locked.

The input $\overline{\text{RSTIN}}$ provides an internal pullup device equalling a resistor of 50 KΩ to 150 KΩ (the minimum reset time must be determined by the lowest value). Simply connecting an external capacitor is sufficient for an automatic power-on reset (see *b*) in figure above). $\overline{\text{RSTIN}}$ may also be connected to the output of other logic gates (see *a*) in figure above). See also section „Bidirectional Reset“ in this case.

Note: A power-on reset requires an active time of two reset sequences (1036 CPU clock cycles) after a stable clock signal is available (about 10...50 ms, depending on the oscillator frequency, to allow the on-chip oscillator to stabilize).

Software Reset

The reset sequence can be triggered at any time via the protected instruction SRST (Software Reset). This instruction can be executed deliberately within a program, e.g. to leave bootstrap loader mode, or upon a hardware trap that reveals a system failure.

Note: A software reset only latches the configuration of the bus interface (SALSEL, CSSEL, WRC, BUSTYP) from PORT0.
If bidirectional reset is enabled, a software reset is executed like a long hardware reset.

Watchdog Timer Reset

When the watchdog timer is not disabled during the initialization or serviced regularly during program execution it will overflow and trigger the reset sequence. Other than hardware and software reset the watchdog reset completes a running external bus cycle. Then the internal reset sequence is started.

Note: A watchdog reset only latches the configuration of the bus interface (SALSEL, CSSEL, WRC, BUSTYP) from PORT0.
If bidirectional reset is enabled a watchdog timer reset is executed like a long hardware reset.
The watchdog reset cannot occur while the C164 is in bootstrap loader mode!

Bidirectional Reset

In a special mode (bidirectional reset) the C164's line RSTIN (normally an input) may be driven active by the chip logic e.g. in order to support external equipment which is required for startup (e.g. flash memory).

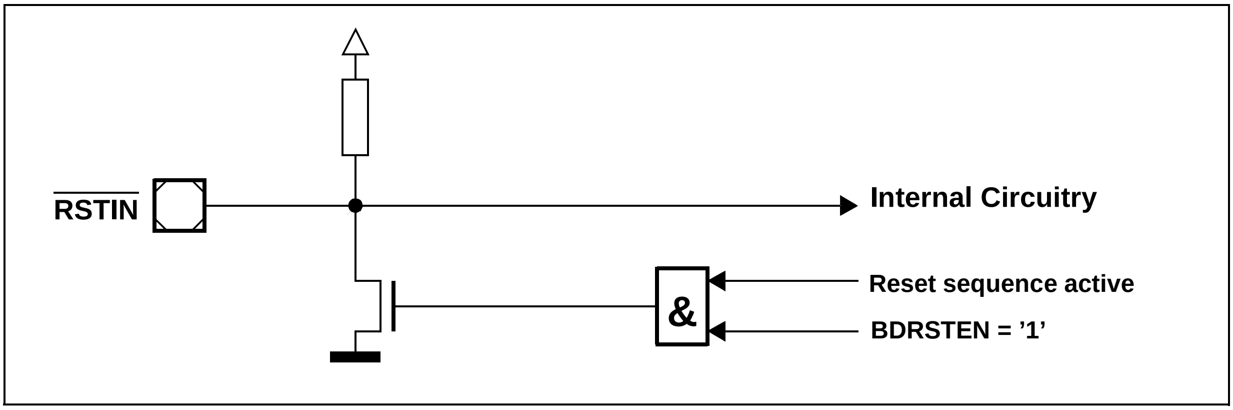


Figure 20-2 Bidirectional Reset Operation

Bidirectional reset reflects internal reset sources (software, watchdog) also to the RSTIN pin and converts short hardware reset pulses to a minimum duration of the internal reset sequence. Bidirectional reset is enabled by setting bit BDRSTEN in register SYSCON and changes RSTIN from a pure input to an open drain IO line. When an internal reset is triggered by the SRST instruction or by a watchdog timer overflow or a low level is applied to the RSTIN line, an internal driver pulls it low for the duration of the internal reset sequence. After that it is released and is then controlled by the external circuitry alone.

The bidirectional reset function is useful in applications where external devices require a defined reset signal but cannot be connected to the C164's RSTOUT signal, e.g. an external flash memory which must come out of reset and deliver code well before RSTOUT can be deactivated via EINIT.

The following behaviour differences must be observed when using the bidirectional reset feature in an application:

- Bit BDRSTEN in register SYSCON cannot be changed after EINIT.
- After a reset bit BDRSTEN is cleared.
- The reset indication flags always indicate a long hardware reset.
- The PORT0 configuration is treated like on a hardware reset. Especially the bootstrap loader may be activated when P0L.4 or RD is low.
- Pin RSTIN may only be connected to ext. reset devices with an open drain output driver.
- A short hardware reset is extended to the duration of the internal reset sequence.

20.2 Status After Reset

After a reset is completed most units of the C164 enter a well-defined default status. This ensures repeatable start conditions and avoids spurious activities after reset.

Watchdog Timer Operation after Reset

The watchdog timer starts running after the internal reset has completed. It will be clocked with the internal system clock divided by 2 (f_{CPU} / 2), and its default reload value is 00_H, so a watchdog timer overflow will occur 131,072 CPU clock cycles ($2 * 2^{16}$) after completion of the internal reset, unless it is disabled, serviced or reprogrammed meanwhile. When the system reset was caused by a watchdog timer overflow, the WDTR (Watchdog Timer Reset Indication) flag in register WDTCON will be set to ’1’. This indicates the cause of the internal reset to the software initialization routine. WDTR is reset to ’0’ by an external hardware reset, by servicing the watchdog timer or after EINIT. After the internal reset has completed, the operation of the watchdog timer can be disabled by the DISWDT (Disable Watchdog Timer) instruction. This instruction has been implemented as a protected instruction. For further security, its execution is only enabled in the time period after a reset until either the SRVWDT (Service Watchdog Timer) or the EINIT instruction has been executed. Thereafter the DISWDT instruction will have no effect.

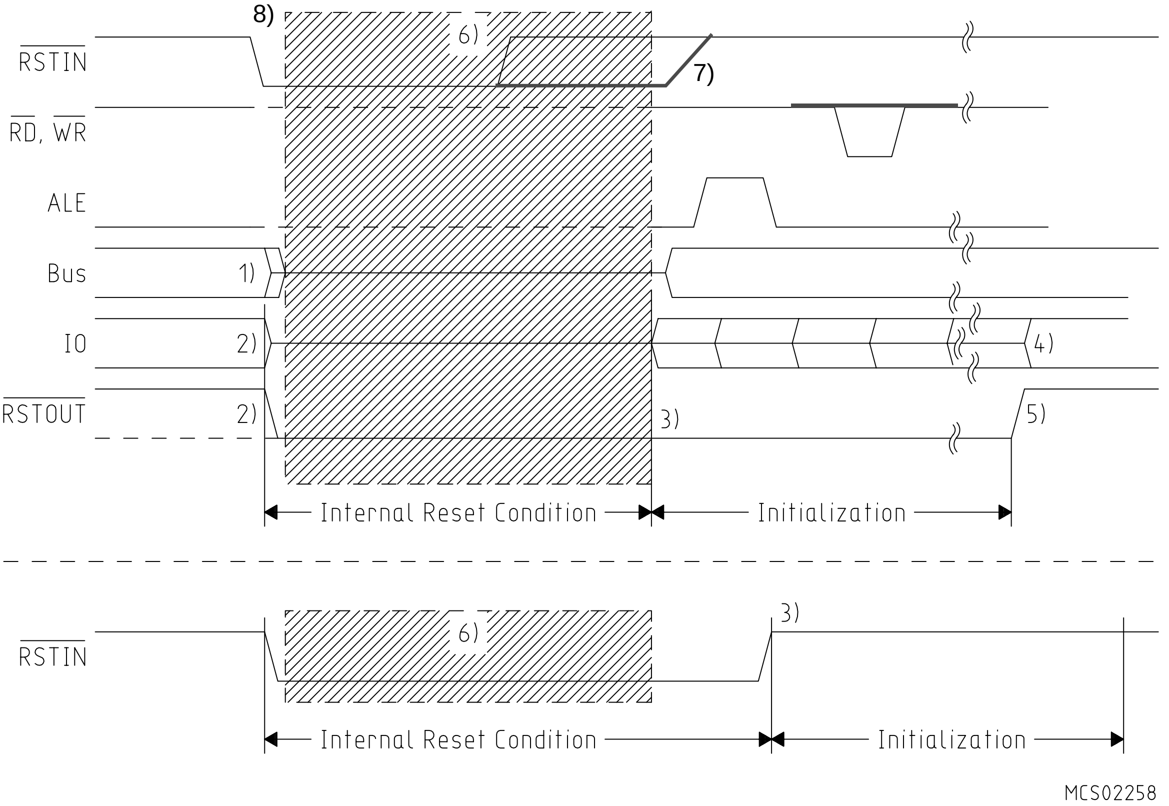
Reset Values for the C164 Registers

During the reset sequence the registers of the C164 are preset with a default value. Most SFRs, including system registers and peripheral control and data registers, are cleared to zero, so all peripherals and the interrupt system are off or idle after reset. A few exceptions to this rule provide a first pre-initialization, which is either fixed or controlled by input pins.

DPP1:	0001_H (points to data page 1)
DPP2:	0002_H (points to data page 2)
DPP3:	0003_H (points to data page 3)
CP:	$FC00_H$
STKUN:	$FC00_H$
STKOV:	$FA00_H$
SP:	$FC00_H$
WDTCON:	$00XX_H$ (value depends on the reset source)
S0RBUF:	XX_H (undefined)
SSCRB:	$XXXX_H$ (undefined)
SYSCON:	$0XX0_H$ (set according to reset configuration)
BUSCON0:	$0XX0_H$ (set according to reset configuration)
RP0H:	XX_H (reset levels of P0H)
ONES:	$FFFF_H$ (fixed value)

The C164's Pins after Reset

After the reset sequence the different groups of pins of the C164 are activated in different ways depending on their function. Bus and control signals are activated immediately after the reset sequence according to the configuration latched from PORT0, so either external accesses can takes place or the external control signals are inactive. The general purpose IO pins remain in input mode (high impedance) until reprogrammed via software (see figure below). The $\overline{\text{RSTOUT}}$ pin remains active (low) until the end of the initialization routine (see description).



When the internal reset condition is extended by $\overline{\text{RSTIN}}$,
the activation of the output signals is delayed until the end of the internal reset condition.

1) Current bus cycle is completed or aborted.
2) Switches asynchronously with $\overline{\text{RSTIN}}$, synchronously upon software or watchdog reset.
3) The reset condition ends here. The C164 starts program execution.
4) Activation of the IO pins is controlled by software.
5) Execution of the EINIT instruction.
6) The shaded area designates the internal reset sequence, which starts after synchronization of $\overline{\text{RSTIN}}$.
7) A short hardware reset is extended until the end of the reset sequence in Bidirectional reset mode.
8) A software or WDT reset activates the $\overline{\text{RSTIN}}$ line in Bidirectional reset mode.

Figure 20-3 Reset Input and Output Signals

Ports and External Bus Configuration during Reset

During the internal reset sequence all of the C164's port pins are configured as inputs by clearing the associated direction registers, and their pin drivers are switched to the high impedance state. This ensures that the C164 and external devices will not try to drive the same pin to different levels. Pin ALE is held low through an internal pulldown, and pins $\overline{\text{RD}}$, $\overline{\text{WR}}$ and $\overline{\text{READY}}$ are held high through internal pullups. Also the pins that can be configured for $\overline{\text{CS}}$ output will be pulled high.

The registers SYSCON and BUSCON0 are initialized according to the configuration selected via PORT0.

When an external start is selected (pin $\overline{\text{EA}}$='0'):

- the Bus Type field (BTYP) in register BUSCON0 is initialized according to P0L.7 and P0L.6
- bit BUSACT0 in register BUSCON0 is set to '1'
- bit ALECTL0 in register BUSCON0 is set to '1'
- bit ROMEN in register SYSCON will be cleared to '0'
- bit BYTDIS in register SYSCON is set according to the data bus width (set if 8-bit)
- bit WRCFG in register SYSCON is set according to pin P0H.0 (WRC)

When an internal start is selected (pin $\overline{\text{EA}}$='1'):

- register BUSCON0 is cleared to 0000_H
- bit ROMEN in register SYSCON will be set to '1'
- bit BYTDIS in register SYSCON is set, i.e. $\overline{\text{BHE}}/\overline{\text{WRH}}$ is disabled
- bit WRCFG in register SYSCON is set according to pin P0H.0 (WRC)

The other bits of register BUSCON0, and the other BUSCON registers are cleared. This default initialization selects the slowest possible external accesses using the configured bus type.

When the internal reset has completed, the configuration of PORT0, PORT1, Port 4, and of the $\overline{\text{BHE}}$ signal (High Byte Enable, alternate function of P3.12) depends on the bus type which was selected during reset. When any of the external bus modes was selected during reset, PORT0 will operate in the selected bus mode. Port 4 will output the selected number of segment address lines (all zero after reset), and will drive the selected number of $\overline{\text{CS}}$ lines ($\overline{\text{CS0}}$ will be '0', while the other active $\overline{\text{CS}}$ lines will be '1'). When no memory accesses above 64 K are to be performed, segmentation may be disabled.

When the on-chip bootstrap loader was activated during reset, pin TxD0 (alternate port function) will be switched to output mode after the reception of the zero byte.

All other pins remain in the high-impedance state until they are changed by software or peripheral operation.

Reset Output Pin

The $\overline{\text{RSTOUT}}$ pin is dedicated to generate a reset signal for the system components besides the controller itself. $\overline{\text{RSTOUT}}$ will be driven active (low) at the begin of any reset sequence (triggered by hardware, the SRST instruction or a watchdog timer overflow). $\overline{\text{RSTOUT}}$ stays active (low) beyond the end of the internal reset sequence until the protected EINIT (End of Initialization) instruction is executed (see figure above). This allows the complete configuration of the controller including its on-chip peripheral units before releasing the reset signal for the external peripherals of the system.

Note: $\overline{\text{RSTOUT}}$ will float as long as pins P0L.0 and P0L.1 select emulation mode or adapt mode.

The Internal RAM after Reset

The contents of the internal RAM are not affected by a system reset. However, after a power-on reset, the contents of the internal RAM are undefined. This implies that the GPRs (R15...R0) and the PEC source and destination pointers (SRCP7...SRCP0, DSTP7...DSTP0) which are mapped into the internal RAM are also unchanged after a warm reset, software reset or watchdog reset, but are undefined after a power-on reset.

The Extension RAM (XRAM) after Reset

The contents of the on-chip extension RAM are not affected by a system reset. However, after a power-on reset, the contents of the XRAM are undefined.

Operation after Reset

After the internal reset condition is removed the C164 fetches the first instruction from the program memory (location 00’0000$_H$ for a standard start). As a rule, this first location holds a branch instruction to the actual initialization routine that may be located anywhere in the address space.

Note: When the Bootstrap Loader Mode was activated during a hardware reset the C164 does not fetch instructions from the program memory.
The standard bootstrap loader expects data via serial interface ASC0.

20.3 Application-Specific Initialization Routine

After a reset the modules of the C164 must be initialized to enable their operation on a given application. This initialization depends on the task the C164 is to fulfill in that application and on some system properties like operating frequency, connected external circuitry, etc.

The following initializations should typically be done, before the C164 is prepared to run the actual application software:

Memory Areas

The external bus interface can be reconfigured after an external reset because register BUSCON0 is initialized to the slowest possible bus cycle configuration. The programmable address windows can be enabled in order to adapt the bus cycle characteristics to different memory areas or peripherals. Also after a single-chip mode reset the external bus interface can be enabled and configured.

The internal program memory (if available) can be enabled and mapped after an external reset in order to use the on-chip resources. After a single-chip mode reset the internal program memory can be remapped or disabled at all in order to utilize external memory (partly or completely).

Programmable program memory can be programmed, e.g. with data received over a serial link.

Note: Initial Flash or OTP programming will rather be done in bootstrap loader mode.

System Stack

The deafult setup for the system stack (size, stackpointer, upper and lower limit registers) can be adjusted to application-specific values. After reset, registers SP and STKUN contain the same reset value 00’FC00$_H$, while register STKOV contains 00’FA00$_H$. With the default reset initialization, 256 words of system stack are available, where the system stack selected by the SP grows downwards from 00’FBFE$_H$.

Note: The interrupt system, which is disabled upon completion of the internal reset, should remain disabled until the SP is initialized.
Traps (incl. $\overline{\text{NMI}}$) may occur, even though the interrupt system is still disabled.

Register Bank

The location of a register bank is defined by the context pointer (CP) and can be adjusted to an application-specific bank before the general purpose registers (GPRs) are used. After reset, register CP contains the value 00’FC00$_H$, i.e. the register bank selected by the CP grows upwards from 00’FC00$_H$.

On-Chip RAM

Based on the application, the user may wish to initialize portions of the internal writable memory (IRAM/XRAM) before normal program operation. Once the register bank has been selected by programming the CP register, the desired portions of the internal memory can easily be initialized via indirect addressing.

Interrupt System

After reset the individual interrupt nodes and the global interrupt system are disabled. In order to enable interrupt requests the nodes must be assigned to their respective interrupt priority levels and be enabled. The vector locations must receive pointers to the respective exception handlers. The interrupt system must globally be enabled by setting bit IEN in register PSW. Care must be taken not to enable the interrupt system before the initialization is complete in order to avoid e.g. the corruption of internal memory locations by stack operations using an uninitialized stack pointer.

Watchdog Timer

After reset the watchdog timer is active and is counting its default period. If the watchdog timer shall remain active the desired period should be programmed by selecting the appropriate prescaler value and reload value. Otherwise the watchdog timer must be disabled before EINIT.

Ports

Generally all ports of the C164 are switched to input after reset. Some pins may be automatically controlled, e.g. bus interface pins for an external start, TxD in Boot mode, etc. Pins that shall be used for general purpose IO must be initialized via software. The required mode (input/output, open drain/push pull, input threshold, etc.) depends on the intended function for a given pin.

Peripherals

After reset the C164's on-chip peripheral modules enter a defined default state (see respective peripheral description) where it is disabled from operation. In order to use a certain peripheral it must be initialized according to its intended operation in the application.

This includes selecting the operating mode (e.g. counter/timer), operating parameters (e.g. baudrate), enabling interface pins (if required), assigning interrupt nodes to the respective priority levels, etc.

After these standard initialization also application-specific actions may be required like asserting certain levels to output pins, sending codes via interfaces, latching input levels, etc.

Termination of Initialization

The software initialization routine should be terminated with the EINIT instruction. This instruction has been implemented as a protected instruction.

The execution of the EINIT instruction...

- disables the action of the DISWDT instruction,
- disables write accesses to reg. SYSCON (all configurations regarding reg. SYSCON (enable CLKOUT, stacksize, etc.) must be selected before the execution of EINIT),
- disables write accesses to registers SYSCON2 and SYSCON3 (further write accesses to SYSCON2 and SYSCON3 can be executed only using a special unlock mechanism),
- clears the reset source detection bits in register WDTCON,
- causes the $\overline{\text{RSTOUT}}$ pin to go high (this signal can be used to indicate the end of the initialization routine and the proper operation of the microcontroller to external hardware).

20.4 System Startup Configuration

Although most of the programmable features of the C164 are selected by software either during the initialization phase or repeatedly during program execution, there are some features that must be selected earlier, because they are used for the first access of the program execution (e.g. internal or external start selected via $\overline{\text{EA}}$).

These configurations are accomplished by latching the logic levels at a number of pins at the end of the internal reset sequence. During reset internal pullup/pulldown devices are active on those lines. They ensure inactive/default levels at pins which are not driven externally. External pulldown/pullup devices may override the default levels in order to select a specific configuration. Many configurations can therefore be coded with a minimum of external circuitry.

Note: The load on those pins that shall be latched for configuration must be small enough for the internal pullup/pulldown device to sustain the default level, or external pullup/pulldown devices must ensure this level.
Those pins whose default level shall be overridden must be pulled low/high externally.
Make sure that the valid target levels are reached until the end of the reset sequence.
There is a specific application note to illustrate this.

20.4.1 System Startup Configuration upon an External Reset

For an external reset ($\overline{EA}$ = '0') the startup configuration uses the pins of PORT0 and pin $\overline{RD}$. The value on the upper byte of PORT0 (P0H) is latched into register RP0H upon reset, the value on the lower byte (P0L) directly influences the BUSCON0 register (bus mode) or the internal control logic of the C164.

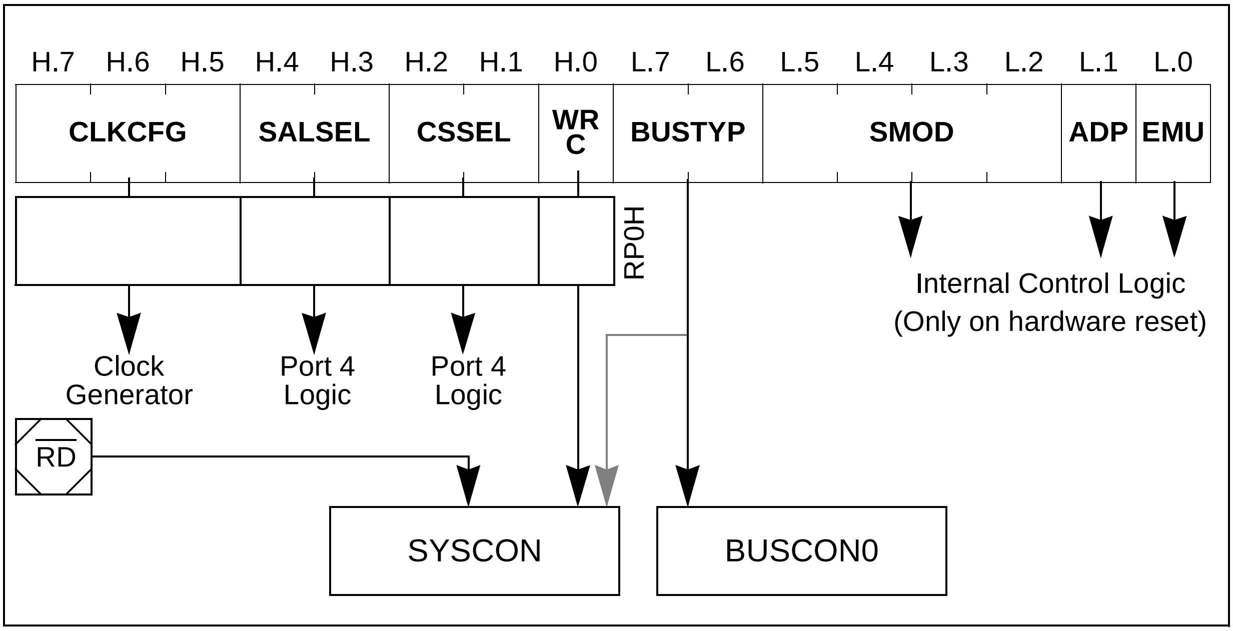


Figure 20-4 PORT0 Configuration during Reset

The pins that control the operation of the internal control logic, the clock configuration, and the reserved pins are evaluated only during a hardware triggered reset sequence. The pins that influence the configuration of the C164 are evaluated during any reset sequence, i.e. also during software and watchdog timer triggered resets.

The configuration via P0H is latched in register RP0H for subsequent evaluation by software. Register RP0H is described in chapter "The External Bus Interface".

The following describes the different selections that are offered for reset configuration. The default modes refer to pins at high level, i.e. without external pulldown devices connected.
Please also consider the note above.

Emulation Mode

Pin P0L.0 (EMU) selects the Emulation Mode, when latched low at the end of reset. This mode is used for special emulation and testing purposes and is of minor use for standard C164 applications, so P0L.0 should be held high.

Emulation mode provides access to integrated XBUS peripherals via the external bus interface pins (direction reversed) of the C164. The CPU and the generic peripherals are disabled, all modules connected via the XBUS are active.

Table 20-1 Emulation Mode Summary

Pin(s)	Function	Notes
Port 4, PORT1	Address input	The segment address lines configured at reset must be driven externally
PORT0	Data input/output	
$\overline{\text{RD}}$, $\overline{\text{WR}}$	Control signal input	
ALE	Unused input	Hold LOW
CLKOUT	CPU clock output	Enabled automatically
$\overline{\text{RSTOUT}}$	Reset input	Drive externally for an XBUS peripheral reset
$\overline{\text{RSTIN}}$	Reset input	Standard reset for complete device
Port 6	Interrupt output	Sends XBUS peripheral interrupt request e.g. to the emulation system

Default: Emulation Mode is off.

Note: In emulation mode pin P0.15 (P0H.7) is inverted, i.e. the configuration '111' would select direct drive in emulation mode.
Emulation mode can only be activated upon an external reset ($\overline{\text{EA}}$ = '0'). Pin P0L.0 is not evaluated upon a single-chip reset ($\overline{\text{EA}}$ = '1').

Adapt Mode

Pin P0L.1 (ADP) selects the Adapt Mode, when latched low at the end of reset. In this mode the C164 goes into a passive state, which is similar to its state during reset. The pins of the C164 float to tristate or are deactivated via internal pullup/pulldown devices, as described for the reset state. In addition also the $\overline{\text{RSTOUT}}$ pin floats to tristate rather than be driven low. The on-chip oscillator and the realtime clock are disabled.

This mode allows switching a C164 that is mounted to a board virtually off, so an emulator may control the board's circuitry, even though the original C164 remains in its place. The original C164 also may resume to control the board after a reset sequence with P0L.1 high. Please note that adapt mode overrides any other configuration via PORT0.

Default: Adapt Mode is off.

Note: When XTAL1 is fed by an external clock generator (while XTAL2 is left open), this clock signal may also be used to drive the emulator device.
However, if a crystal is used, the emulator device's oscillator can use this crystal only, if at least XTAL2 of the original device is disconnected from the circuitry (the output XTAL2 will be driven high in Adapt Mode).
Adapt mode can only be activated upon an external reset ($\overline{\text{EA}}$ = '0'). Pin P0L.1 is not evaluated upon a single-chip reset ($\overline{\text{EA}}$ = '1').

Special Operation Modes

Pins P0L.5 to P0L.2 (SMOD) select special operation modes of the C164 during reset (see table below). Make sure to only select valid configurations in order to ensure proper operation of the C164.

Table 20-2 Definition of Special Modes for Reset Configuration

P0.5-2 (P0L.5-2)	Special Mode	Notes
1 1 1 1	Normal Start	Default configuration. Begin of execution as defined via pin $\overline{\text{EA}}$.
1 1 1 0	Reserved	Do not select this configuration!
1 1 0 1	Reserved	Do not select this configuration!
1 1 0 0	Reserved	Do not select this configuration!
1 0 1 1	Standard Bootstrap Loader	Load an initial boot routine of 32 bytes via interface ASC0.
1 0 1 0	Reserved	Do not select this configuration!
1 0 0 1	Alternate Boot	Operation not yet defined. Do not use!
1 0 0 0	Reserved	Do not select this configuration!
0 1 1 1	No emulation mode: Alternate Start	Operation not yet defined. Do not use!
0 1 1 0	Reserved	Do not select this configuration!
0 1 0 1	Reserved	Do not select this configuration!
0 1 0 0	Reserved	Do not select this configuration!
0 0 X X	Reserved	Do not select this configuration!

The on-chip Bootstrap Loader allows moving the start code into the internal RAM of the C164 via the serial interface ASC0. The C164 will remain in bootstrap loader mode until a hardware reset not selecting BSL mode or a software reset.

Default: The C164 starts fetching code from location 00’0000$_H$, the bootstrap loader is off.

External Bus Type

Pins P0L.7 and P0L.6 (BUSTYP) select the external bus type during reset, if an external start is selected via pin $\overline{\text{EA}}$. This allows the configuration of the external bus interface of the C164 even for the first code fetch after reset. The two bits are copied into bit field BTYP of register BUSCON0. P0L.7 controls the data bus width, while P0L.6 controls the address output (multiplexed or demultiplexed). This bit field may be changed via software after reset, if required.

Table 20-3 Configuration of External Bus Type

P0L.7-6 (BTYP) Encoding	External Data Bus Width	External Address Bus Mode
0 0	8-bit Data	Demultiplexed Addresses
0 1	8-bit Data	Multiplexed Addresses
1 0	16-bit Data	Demultiplexed Addresses
1 1	16-bit Data	Multiplexed Addresses

PORT0 and PORT1 are automatically switched to the selected bus mode. In multiplexed bus modes PORT0 drives both the 16-bit intra-segment address and the output data, while PORT1 remains in high impedance state as long as no demultiplexed bus is selected via one of the BUSCON registers. In demultiplexed bus modes PORT1 drives the 16-bit intra-segment address, while PORT0 or P0L (according to the selected data bus width) drives the output data.
For a 16-bit data bus $\overline{\text{BHE}}$ is automatically enabled, for an 8-bit data bus $\overline{\text{BHE}}$ is disabled via bit BYTDIS in register SYSCON.

Default: 16-bit data bus with multiplexed addresses.

Note: If an internal start is selected via pin $\overline{\text{EA}}$, these two pins are disregarded and bit field BTYP of register BUSCON0 is cleared.

Write Configuration

Pin P0H.0 (WRC) selects the initial operation of the control pins $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ during reset. When high, this pin selects the standard function, i.e. $\overline{\text{WR}}$ control and $\overline{\text{BHE}}$. When low, it selects the alternate configuration, i.e. $\overline{\text{WRH}}$ and $\overline{\text{WRL}}$. Thus even the first access after a reset can go to a memory controlled via $\overline{\text{WRH}}$ and $\overline{\text{WRL}}$. This bit is latched in register RP0H and its inverted value is copied into bit WRCFG in register SYSCON.

Default: Standard function ($\overline{\text{WR}}$ control and $\overline{\text{BHE}}$).

Chip Select Lines

Pins P0H.2 and P0H.1 (CSSEL) define the number of active chip select signals during reset. This allows the selection which pins of Port 4 drive external $\overline{\text{CS}}$ signals and which are used for general purpose IO. The two bits are latched in register RP0H.

Table 20-4 Configuration of Chip Select Lines

P0H.2-1 (CSSEL)	Chip Select Lines	Note
1 1	Four: $\overline{\text{CS3}}$...$\overline{\text{CS0}}$	Default without pull-downs
1 0	None	
0 1	Two: $\overline{\text{CS1}}$...$\overline{\text{CS0}}$	
0 0	Three: $\overline{\text{CS2}}$...$\overline{\text{CS0}}$	

Default: All 4 chip select lines active ($\overline{\text{CS3}}$...$\overline{\text{CS0}}$).

Note: The selected number of $\overline{\text{CS}}$ signals can be changed via software after reset (see section 20.4.2 on page 20).

Segment Address Lines

Pins P0H.4 and P0H.3 (SALSEL) define the number of active segment address lines during reset. This allows the selection which pins of Port 4 drive address lines and which are used for general purpose IO. The two bits are latched in register RP0H. Depending on the system architecture the required address space is chosen and accessible right from the start, so the initialization routine can directly access all locations without prior programming. The required pins of Port 4 are automatically switched to address output mode.

Table 20-5 Configuration of Segment Address Lines

P0H.4-3 (SALSEL)	Segment Address Lines	Directly accessible A. Space
1 1	Two: A17...A16	256 KByte (Default without pull-downs)
1 0	Six: A21...A16	4 MByte (Maximum)
0 1	None	64 KByte (Minimum)
0 0	Four: A19...A16	1 MByte

Even if not all segment address lines are enabled on Port 4, the C164 internally uses its complete 24-bit addressing mechanism. This allows the restriction of the width of the effective address bus, while still deriving $\overline{\text{CS}}$ signals from the complete addresses.

Default: 2-bit segment address (A17...A16) allowing access to 256 KByte.

Note: The selected number of segment address lines can be changed via software after reset. (See section 20.4.2 on page 20).

Clock Generation Control

Pins P0H.7, P0H.6 and P0H.5 (CLKCFG) select the basic clock generation mode during reset. The oscillator clock either directly feeds the CPU and peripherals (direct drive), it is divided by 2 or it is fed to the on-chip PLL which then provides the CPU clock signal (selectable multiple of the oscillator frequency, i.e. the input frequency). These bits are latched in register RP0H.

Table 20-6 C164 Clock Generation Modes

P0.15-13 (P0H.7-5)	CPU Frequency $f_{CPU} = f_{OSC} * F$	External Clock Input Range [1)]	Notes
1 1 1	$f_{OSC} * 4$	2.5 to 6.25 MHz	Default configuration
1 1 0	$f_{OSC} * 3$	3.33 to 8.33 MHz	
1 0 1	$f_{OSC} * 2$	5 to 12.5 MHz	
1 0 0	$f_{OSC} * 5$	2 to 5 MHz	
0 1 1	$f_{OSC} * 1$	1 to 25 MHz	Direct drive [2)]
0 1 0	$f_{OSC} * 1.5$	6.66 to 16.6 MHz	
0 0 1	$f_{OSC} / 2$	2 to 50 MHz	CPU clock via prescaler
0 0 0	$f_{OSC} * 2.5$	4 to 10 MHz	

[1)] The external clock input range refers to a CPU clock range of 10...25 MHz.

[2)] The maximum frequency depends on the duty cycle of the external clock signal.
In emulation mode pin P0.15 (P0H.7) is inverted, i.e. the configuration '111' would select direct drive in emulation mode.

Default: On-chip PLL is active with a factor of 1:4.

Watch the different requirements for frequency and duty cycle of the oscillator input clock for the possible selections.

Oscillator Watchdog Control

The on-chip oscillator watchdog (OWD) may be disabled via hardware by (externally) pulling the $\overline{\text{RD}}$ line low upon a reset, similar to the standard reset configuration via PORT0. At the end of any reset bit OWDDIS in register SYSCON reflects the inverted level of pin $\overline{\text{RD}}$ at that time. The software may again enable the oscillator watchdog by clearing bit OWDDIS before the execution of EINIT.

Note: If direct drive or prescaler operation is selected as basic clock generation mode (see above) the PLL is switched off whenever bit OWDDIS is set (via software or via hardware configuration).

20.4.2 System Startup Configuration upon a Single-Chip Mode Reset

For a single-chip mode reset (indicated by $\overline{\text{EA}}$ = '1') the configuration via PORT0 is replaced by a fixed configuration value. In this case PORT0 needs no external circuitry (pullups/pulldowns) and also the internal configuration pullups are not activated.

The necessary startup modes are configured via pins $\overline{\text{RD}}$ and ALE.

This fixed default configuration is activated after each Long Hardware Reset and selects a safe worst-case configuration. The initialization software can then modify these parameters and select the intended configuration for a given application. The table below lists the respective default configuration values which are selected, and the bitfields that permit software modification.

Table 20-7 Default Configuration for Single-Chip Mode Reset

Configuration Parameter	Default Value	Ext.Conf.[1]	Software Acc.[2]
CLKCFG: Generation mode of basic clock	001_B Prescaler operation, i.e. $f_{CPU} = f_{OSC} / 2$	P0.15-13	RSTCON.15-13
SALSEL: Number of active segment address lines	01_B No segment address lines	P0.12-11	RSTCON.12-11
CSSEL: Number of active $\overline{\text{CS}}$ lines	10_B No chip select lines	P0.10-9	RSTCON.10-9
WRC: Write signal encoding	SYSCON.WRCFG = 0_B i.e. $\overline{\text{WR}}$ and $\overline{\text{BHE}}$	P0.8	SYSCON.WRCFG
BTYP: Default bustype (BUSCON0)	BUSCON0.BTYP = 11_B i.e. 16-bit MUX bus	P0.7-6	BUSCON0.BTYP
SMOD: Special modes (start/boot modes)	Startup modes selected via pins $\overline{\text{RD}}$ and ALE	P0.5-2	-----
ADP: Adapt mode	Not possible	P0.1	-----
EMU: Emulation mode	Not possible	P0.0	-----
OWD disable	SYSCON.OWDDIS = 0_B i.e. OWD is active	$\overline{\text{RD}}$	SYSCON. OWDDIS

[1] Refers to the configuration pins which are replaced by the default values.

[2] Software can modify the default values via these bitfields.

Note: The indicated default values result in the reset value XX2B$_H$ for register RP0H.

Single-Chip Startup Modes

The startup mode (operation after reset) of the C164 can be configured during reset. In single-chip mode this configuration is selected via pins $\overline{\text{RD}}$ and ALE.

Pin $\overline{\text{RD}}$ selects start or boot mode (instead of OWD control), pin ALE selects one of two alternatives in each case.

Table 20-8 Startup Mode Configuration in Single-Chip Reset Mode

$\overline{\text{RD}}$	ALE	Startup Mode	Notes
1	0	Standard Start	Execution starts at user memory location 00’0000$_H$.
1	1	Alternate Start	Operation not yet defined. Do not use!
0	0	Standard Bootstrap Loader	Load 32 bytes via ASC0.
0	1	Alternate Boot Mode	Operation not yet defined. Do not use!

20.5 System Configuration via Software

The system configuration which is selected via hardware after reset (latched pin levels or default value) can be changed via software by executing a specific code sequence. The respective control bits are located within registers SYSCON, BUSCONx, and RSTCON. Register SYSCON can only be modified before the execution of instruction EINIT, while registers BUSCONx and RSTCON (using the specific sequence) can be modified repeatedly at any time.

The clock generation mode (CLKCFG), the segment address width (SALSEL), and the number of chip select lines (CSSEL) are controlled by register RP0H. RP0H is initialized according to the selected reset mode (pins or default). The respective configuration bitfields can be copied from register RSTCON upon entering Slow Down Divider mode if enabled by bit SUE = ’1’.

The following steps must be taken to change the current configuration (see also SW example):

- Write intended configuration value to RSTCON
- Enter SDD mode
- Return to basic clock mode

```
CHANGE_CLOCK_CONFIGURATION:      ;Note: "RSTCON" is a mem address, no SFR
MOV R15,      #11100001xxxxxxxxB ;Load a GPR with the target value
MOV RSTCON,   R15                ;Enable update with PLL factor 4
EXTR          #2                 ;ESFR access
MOV SYSCON2, #0500H              ;SDD mode, PLL on, factor 2
                                 ;RSTCON.15-9 is copied to RP0H.15-9
MOV SYSCON2, #0400H              ;Switch to basic clock mode
                                 ;System now runs on PLL with factor 4
```

Note: This software example assumes execution before EINIT. Otherwise the unlock sequence has to be executed prior to each access to RSTCON/SYSCON2.

Entering SDD mode temporarily ensures a correct clock signal synchronization in cases where the clock generation mode (e.g. PLL factor) is changed.

Software modification of system configuration values is protected by the following features:

- SYSCON is locked after EINIT
- RSTCON requires the unlock sequence after EINIT
- Copying RSTCON to RP0H must be explicitly enabled by setting bit SUE

Note: RSTCON is write protected after the execution of EINIT unless it is released via the unlock sequence.
RSTCON can only be accessed via its long (mem) address.

RSTCON
Reset Control Register **mem ($F1E0_H$/$--_H$)** **Reset value: $00XX_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLKCFG			**SALSEL**		**CSSEL**		**SUE**	**-**	**-**	**-**	**-**	**-**	**-**	**RSTLEN**	
rw			rw		rw		rw	-	-	-	-	-	-	rw	

Bit	Function
RSTLEN	**Reset Length Control** (duration of the next reset sequence to occur) [1] 00: 1024 TCL: standard duration, corresponds to all other derivatives without control function 01: 2048 TCL: extended duration, may be useful e.g. to provide additional settling for external configuration signals at high CPU clock frequencies 10: Reserved 11: Reserved
SUE	**Software Update Enable** 0: Configuration cannot be changed via software 1: Software update of configuration is enabled
CSSEL	**Chip Select Line Selection** (Number of active $\overline{CS}$ outputs) 00: 3 $\overline{CS}$ lines: $\overline{CS2}$...$\overline{CS0}$ 01: 2 $\overline{CS}$ lines: $\overline{CS1}$...$\overline{CS0}$ 10: No $\overline{CS}$ lines at all 11: all $\overline{CS}$ lines: $\overline{CSx}$...$\overline{CS0}$
SALSEL	**Segment Address Line Selection** (Number of active seg.addr. outputs) 00: 4-bit segment address: A19...A16 01: No segment address lines at all 10: full segment address: Axx...A16 11: 2-bit segment address: A17...A16
CLKCFG	**Clock Generation Mode Configuration** These pins define the clock generation mode, i.e. the mechanism how the the internal CPU clock is generated from the externally applied (XTAL1) input clock. 000: PLL (*f* * 2.5) 100: PLL (*f* * 5) 001: Prescaler (*f* / 2) 101: PLL (*f* * 2) 010: PLL (*f* * 1.5) 110: PLL (*f* * 3) 011: Direct Drive (*f* = *f*) 111: PLL (*f* * 4)

[1] RSTLEN is always valid for the **next** reset sequence. An initial power up reset, however, is expected to last considerably longer than any configurable reset sequence.

21 Power Management

For an increasing number of microcontroller based systems it is an important objective to reduce the power consumption of the system as much as possible. A contradictory objective is, however, to reach a certain level of system performance. Besides optimization of design and technology a microcontroller's power consumption can generally be reduced by lowering its operating frequency and/or by reducing the circuitry that is clocked. The architecture of the C164 provides three major means of reducing its power consumption (see figure below) under software control:

- Reduction of the CPU frequency for Slow Down operation (Flexible Clock Generation Management)
- Selection of the active peripheral modules (Flexible Peripheral Management)
- Special operating modes to deactivate CPU, ports and control logic (Idle, Sleep, Power Down)

This enables the application (i.e. the programmer) to choose the optimum constellation for each operating condition, so the power consumption can be adapted to conditions like maximum performance, partial performance, intermittend operation or standby.

Intermittend operation (i.e. alternating phases of high performance and power saving) is supported by the cyclic interrupt generation mode of the on-chip RTC (real time clock).

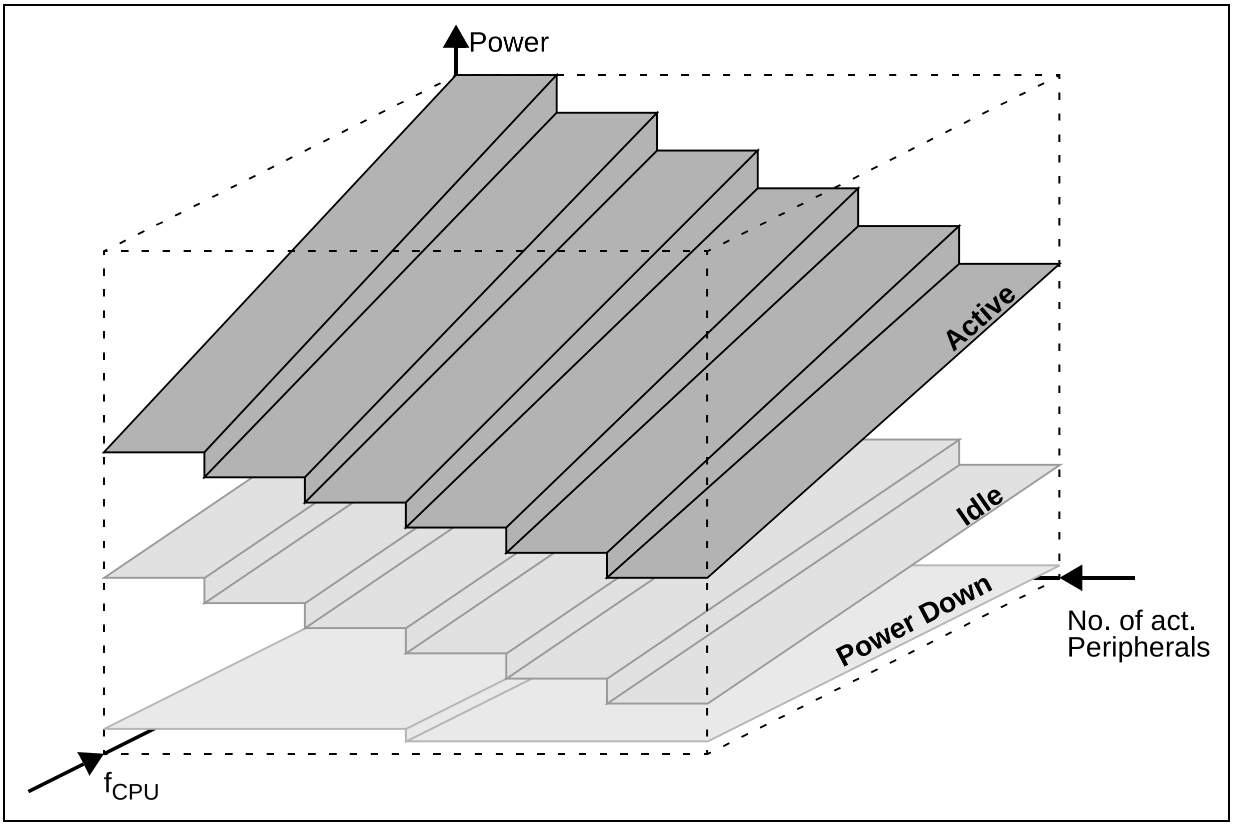


Figure 21-1 Power Reduction Possibilities

These three means described above can be applied independent from each other and thus provide a maximum of flexibility for each application.

For the basic power reduction modes (Idle, Power Down) there are dedicated instructions, while special registers control clock generation (SYSCON2) and peripheral management (SYSCON3).

Three different general power reduction modes with different levels of power reduction have been implemented in the C164, which may be entered under software control.

In **Idle mode** the CPU is stopped, while the (enabled) peripherals continue their operation. Idle mode can be terminated by any reset or interrupt request.

In **Sleep mode** both the CPU and the peripherals are stopped. The real time clock and its selected oscillator may optionally be kept running. Sleep mode can be terminated by any reset or interrupt request (mainly hardware requests, stopped peripherals cannot generate interrupt requests).

In **Power Down mode** both the CPU and the peripherals are stopped. The real time clock and its selected oscillator may optionally be kept running. Power Down mode can only be terminated by a hardware reset.

Note: All external bus actions are completed before a power saving mode is entered.

In addition the power management selects the current CPU frequency and controls which peripherals are active.

During **Slow Down operation** the basic clock generation path is bypassed and the CPU clock is generated via the programmable Slow Down Divider (SDD) from the selected oscillator clock signal.

Peripheral Management disables and enables the on-chip peripheral modules independently, reducing the amount of clocked circuitry including the respective clock drivers.

21.1 Idle Mode

The power consumption of the C164 microcontroller can be decreased by entering Idle mode. In this mode all enabled peripherals, **including** the watchdog timer, continue to operate normally, only the CPU operation is halted and the on-chip memory modules are disabled.

Note: Peripherals that have been disabled via software also remain disabled after entering Idle mode, of course.

Idle mode is entered after the IDLE instruction has been executed and the instruction before the IDLE instruction has been completed (bitfield SLEEPCON in register SYSCON1 must be 00_B). To prevent unintentional entry into Idle mode, the IDLE instruction has been implemented as a protected 32-bit instruction.

Idle mode is terminated by interrupt requests from any enabled interrupt source whose individual Interrupt Enable flag was set before the Idle mode was entered, regardless of bit IEN.

For a request selected for CPU interrupt service the associated interrupt service routine is entered if the priority level of the requesting source is higher than the current CPU priority and the interrupt system is globally enabled. After the RETI (Return from Interrupt) instruction of the interrupt service routine is executed the CPU continues executing the program with the instruction following the IDLE instruction. Otherwise, if the interrupt request cannot be serviced because of a too low priority or a globally disabled interrupt system the CPU immediately resumes normal program execution with the instruction following the IDLE instruction.

For a request which was programmed for PEC service a PEC data transfer is performed if the priority level of this request is higher than the current CPU priority and the interrupt system is globally enabled. After the PEC data transfer has been completed the CPU remains in Idle mode. Otherwise, if the PEC request cannot be serviced because of a too low priority or a globally disabled interrupt system the CPU does not remain in Idle mode but continues program execution with the instruction following the IDLE instruction.

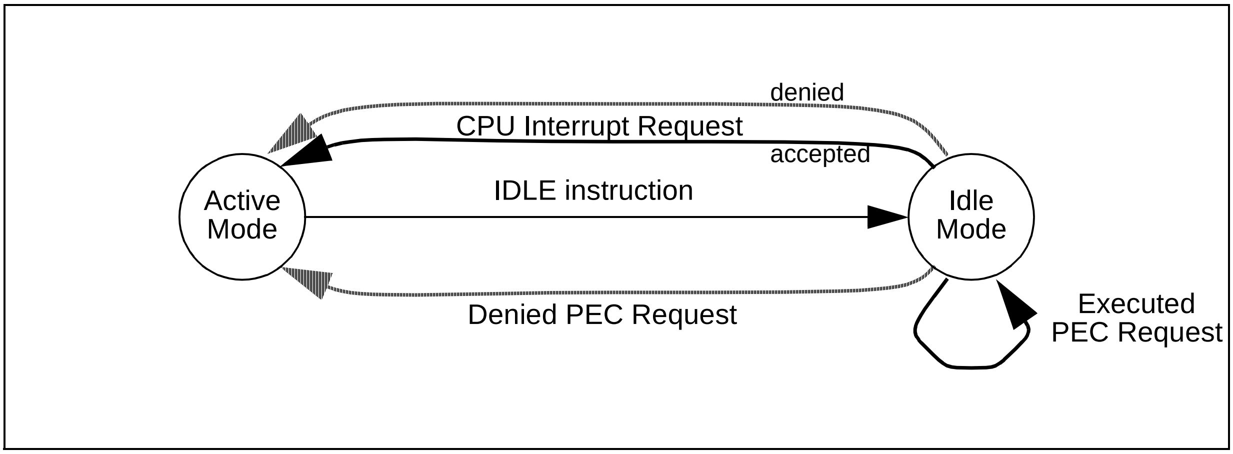


Figure 21-2 Transitions between Idle mode and Active mode

Idle mode can also be terminated by a Non-Maskable Interrupt, i.e. a high to low transition on the $\overline{\text{NMI}}$ pin. After Idle mode has been terminated by an interrupt or NMI request, the interrupt system performs a round of prioritization to determine the highest priority request. In the case of an NMI request, the NMI trap will always be entered.

Any interrupt request whose individual Interrupt Enable flag was set before Idle mode was entered will terminate Idle mode regardless of the current CPU priority. The CPU will **not** go back into Idle mode when a CPU interrupt request is detected, even when the interrupt was not serviced because of a higher CPU priority or a globally disabled interrupt system (IEN='0'). The CPU will **only** go back into Idle mode when the interrupt system is globally enabled (IEN='1') **and** a PEC service on a priority level higher than the current CPU level is requested and executed.

Note: An interrupt request which is individually enabled and assigned to priority level 0 will terminate Idle mode. The associated interrupt vector will not be accessed, however.

The watchdog timer may be used to monitor the Idle mode: an internal reset will be generated if no interrupt or NMI request occurs before the watchdog timer overflows. To prevent the watchdog timer from overflowing during Idle mode it must be programmed to a reasonable time interval before Idle mode is entered.

21.2 Sleep Mode

To further reduce the power consumption the microcontroller can be switched to Sleep mode. Clocking of all internal blocks is stopped (RTC and selected oscillator optionally), the contents of the internal RAM, however, are preserved through the voltage supplied via the V_{DD} pins. The watchdog timer is stopped in Sleep mode.

Sleep mode is selected via bitfield SLEEPCON in register SYSCON1 and is entered after the IDLE instruction has been executed and the instruction before the IDLE instruction has been completed.

Sleep mode is terminated by interrupt requests from any enabled interrupt source whose individual Interrupt Enable flag was set before the Idle mode was entered, regardless of bit IEN. Mainly these are external interrupts and the RTC (if running).

Note: The receive lines of serial interfaces may be internally routed to external interrupt inputs (see EXISEL). All peripherals except for the RTC are stopped and hence cannot generate an interrupt request.

The realtime clock (RTC) can be kept running in Sleep mode in order to maintain a valid system time as long as the supply voltage is applied. This enables a system to determine the current time and the duration of the period while it was down (by comparing the current time with a timestamp stored when Sleep mode was entered). The supply current in this case remains well below 1 mA.

During Sleep mode the voltage at the V_{DD} pins can be lowered to 2.7 V while the RTC and its selected oscillator will still keep on running and the contents of the internal RAM will still be preserved.

When the RTC (and oscillator) is disabled the internal RAM is preserved down to a voltage of 2.5 V.

Note: When the RTC remains active in Sleep mode also the oscillator which generates the RTC clock signal will keep on running, of course.
If the supply voltage is reduced the specified maximum CPU clock frequency for this case must be respected.
For wakeup (input edge recognition and CPU start) the power must be within the specified limits, however.

The total power consumption in Sleep mode depends on the active circuitry (i.e. RTC on or off) and on the current that flows through the port drivers. Individual port drivers can be disabled simply by configuring them for input.

The bus interface pins can be separately disabled by releasing the external bus (disable all address windows by clearing the BUSACT bits) and switching the ports to input (if necessary). Of course the required software in this case must be executed from internal memory.

SYSCON1
System Control Register 1 **ESFR (F1DC$_H$/EE$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**SLEEPCON**	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	rw	

Bit	Function
SLEEPCON	**SLEEP Mode Configuration** (mode entered upon the IDLE instruction) 00: Normal IDLE mode 01: SLEEP mode, with RTC running 10: Reserved. 11: SLEEP mode, with RTC and oscillator stopped

Note: SYSCON1 is write protected after the execution of EINIT unless it is released via the unlock sequence.

21.3 Power Down Mode

The microcontroller can be switched to Power Down mode which reduces the power consumption to a minimum. Clocking of all internal blocks is stopped (RTC and selected oscillator optionally), the contents of the internal RAM, however, are preserved through the voltage supplied via the V_{DD} pins. The watchdog timer is stopped in Power Down mode. This mode can only be terminated by an external hardware reset, i.e. by asserting a low level on the $\overline{\text{RSTIN}}$ pin. This reset will initialize all SFRs and ports to their default state, but will not change the contents of the internal RAM.

There are two levels of protection against unintentionally entering Power Down mode. First, the PWRDN (Power Down) instruction which is used to enter this mode has been implemented as a protected 32-bit instruction. Second, this instruction is effective **only** if the $\overline{\text{NMI}}$ (Non Maskable Interrupt) pin is externally pulled low while the PWRDN instruction is executed. The microcontroller will enter Power Down mode after the PWRDN instruction has completed.

This feature can be used in conjunction with an external power failure signal which pulls the $\overline{\text{NMI}}$ pin low when a power failure is imminent. The microcontroller will enter the NMI trap routine which can save the internal state into RAM. After the internal state has been saved, the trap routine may then execute the PWRDN instruction. If the $\overline{\text{NMI}}$ pin is still low at this time, Power Down mode will be entered, otherwise program execution continues.

The initialization routine (executed upon reset) can check the reset identification flags in register WDTCON to determine whether the controller was initially switched on, or whether it was properly restarted from Power Down mode.

The realtime clock (RTC) can be kept running in Power Down mode in order to maintain a valid system time as long as the supply voltage is applied. This enables a system to determine the current time and the duration of the period while it was down (by comparing the current time with a timestamp stored when Power Down mode was entered). The supply current in this case remains well below 1 mA.

During power down the voltage at the V_{DD} pins can be lowered to 2.7 V while the RTC and its selected oscillator will still keep on running and the contents of the internal RAM will still be preserved.
When the RTC (and oscillator) is disabled the internal RAM is preserved down to a voltage of 2.5 V.

Note: When the RTC remains active in Power Down mode also the oscillator which generates the RTC clock signal will keep on running, of course.
If the supply voltage is reduced the specified maximum CPU clock frequency for this case must be respected.

The total power consumption in Power Down mode depends on the active circuitry (i.e. RTC on or off) and on the current that flows through the port drivers. To minimize the consumed current the RTC and/or all pin drivers can be disabled (pins switched to tristate) via a central control bitfield in register SYSCON2. If an application requires one or more port drivers to remain active even in Power Down mode also individual port drivers can be disabled simply by configuring them for input.

The bus interface pins can be separately disabled by releasing the external bus (disable all address windows by clearing the BUSACT bits) and switching the ports to input (if necessary). Of course the required software in this case must be executed from internal memory.

21.3.1 Status of Output Pins during Power Reduction Modes

During Idle mode the CPU clocks are turned off, while all peripherals continue their operation in the normal way. Therefore all ports pins, which are configured as general purpose output pins, output the last data value which was written to their port output latches. If the alternate output function of a port pin is used by a peripheral, the state of the pin is determined by the operation of the peripheral.

Port pins which are used for bus control functions go into that state which represents the inactive state of the respective function (e.g. $\overline{\text{WR}}$), or to a defined state which is based on the last bus access (e.g. $\overline{\text{BHE}}$). Port pins which are used as external address/data bus hold the address/data which was output during the last external memory access before entry into Idle mode under the following conditions:

P0H outputs the high byte of the last address if a multiplexed bus mode with 8-bit data bus is used, otherwise P0H is floating. P0L is always floating in Idle mode.

PORT1 outputs the lower 16 bits of the last address if a demultiplexed bus mode is used, otherwise the output pins of PORT1 represent the port latch data.

Port 4 outputs the segment address for the last access on those pins that were selected during reset, otherwise the output pins of Port 4 represent the port latch data.

During Sleep mode the oscillator (except for RTC operation) and the clocks to the CPU and to the peripherals are turned off. Like in Idle mode, all port pins which are configured as general purpose output pins output the last data value which was written to their port output latches.

When the alternate output function of a port pin is used by a peripheral the state of this pin is determined by the last action of the peripheral before the clocks were switched off.

During Power Down mode the oscillator (except for RTC operation) and the clocks to the CPU and to the peripherals are turned off. Like in Idle mode, all port pins which are configured as general purpose output pins output the last data value which was written to their port output latches.

When the alternate output function of a port pin is used by a peripheral the state of this pin is determined by the last action of the peripheral before the clocks were switched off.

Note: All pin drivers can be switched off by selecting the general port disable function prior to entering Power Down mode.
When the supply voltage is lowered in Power Down mode the high voltage of output pins will decrease accordingly.

Table 21-1 State of C164 Output Pins during Idle and Power Down mode.

C164 Output Pin(s)	**External Bus Enabled**		**No External Bus**	
	Idle Mode	**Sleep and Power Down**	**Idle Mode**	**Sleep and Power Down**
CLKOUT	Active (toggling)	High	Active (toggling)	High
FOUT	Active (toggling)	Hold (high or low)	Active (toggling)	Hold (high or low)
ALE	Low		Low	
$\overline{RD}$, $\overline{WR}$	High		High	
P0L	Floating		Port Latch Data	
P0H	A15...A8 1) / Float		Port Latch Data	
PORT1	Last Address 2) / Port Latch Data		Port Latch Data	
Port 4	Port Latch Data / Last segment		Port Latch Data	
$\overline{BHE}$	Last value		Port Latch Data	
$\overline{CSx}$	Last value 3)		Port Latch Data	
$\overline{RSTOUT}$	High if EINIT was executed before entering Idle or Power Down mode, Low otherwise.			
Other Port Output Pins	Port Latch Data / Alternate Function			

1) For multiplexed buses with 8-bit data bus.

2) For demultiplexed buses.

3) The $\overline{CS}$ signal that corresponds to the last address remains active (low), all other enabled $\overline{CS}$ signals remain inactive (high). By accessing an on-chip X-Periperal prior to entering a power save mode all external $\overline{CS}$ signals can be deactivated.

21.4 Slow Down Operation

A separate clock path can be selected for Slow Down operation bypassing the basic clock path used for standard operation. The programmable Slow Down Divider (SDD) divides the oscillator frequency by a factor of 1...32 which is specified via bitfield CLKREL in register SYSCON2 (factor = <CLKREL>+1). When bitfield CLKREL is written during SDD operation the reload counter will output one more clock pulse with the „old“ frequency in order to resynchronize internally before generating the „new“ frequency.

If direct drive mode is configured clock signal f_{DD} is directly fed to f_{CPU}, if prescaler mode is configured clock signal f_{DD} is additionally divided by 2:1 to generate f_{CPU} (see examples below).

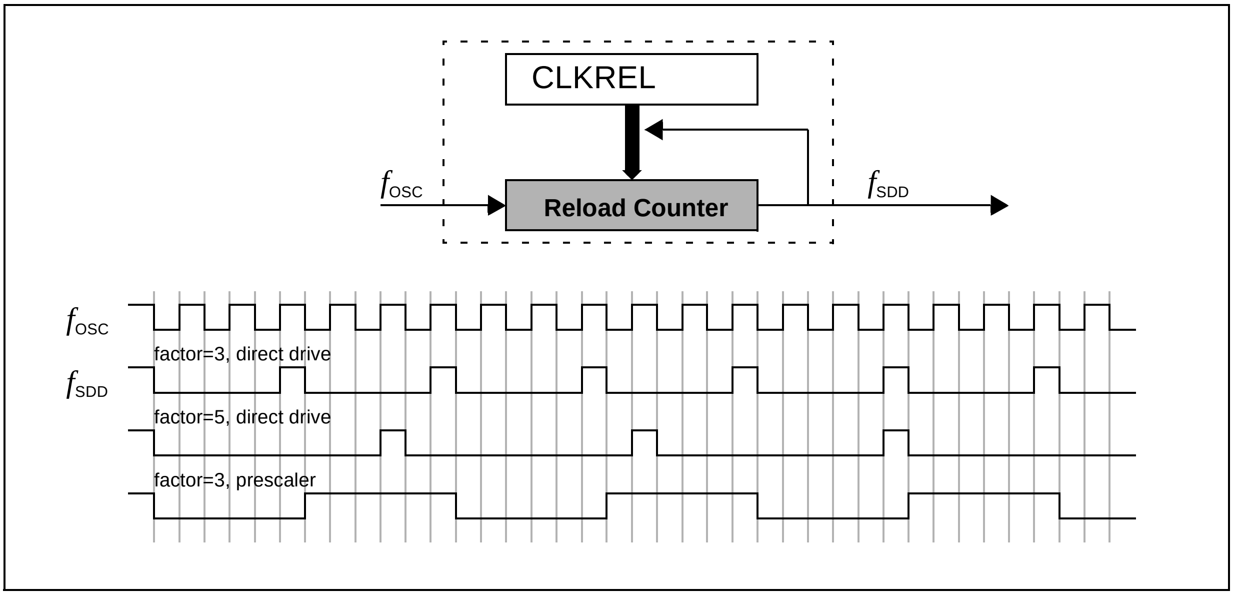


Figure 21-3 Slow Down Divider Operation

Using e.g. a 5 MHz input clock the on-chip logic may be run at a frequency down to 156.25 KHz (or 78 KHz) without an external hardware change. An implemented PLL may be switched off in this case or kept running, depending on the requirements of the application (see table below).

Note: During Slow Down operation the whole device (including bus interface and generation of signals CLKOUT or FOUT) is clocked with the SDD clock (see figure above).

Table 21-2 PLL Operation (if available) in Slow Down Mode

	Advantage	**Disadvantage**	**Oscillator Watchdog**
PLL running	Fast switching back to basic clock source	PLL adds to power consumption	Active if not disabled via bit OWDDIS
PLL off	PLL causes no additional power consumption	PLL must lock before switching back to the basic clock source (if the PLL is the basic clock source)	Disabled

All these clock options are selected via bitfield CLKCON in register SYSCON2. A state machine controls the switching mechanism itself and ensures a continuous and glitch-free clock signal to the on-chip logic. This is especially important when switching back to PLL frequency when the PLL has temporarily been switched off. In this case the clock source can be switched back either automatically as soon as the PLL is locked again (indicated by bit CLKLOCK in register SYSCON2), or manually, i.e. under software control, after bit CLKLOCK has become ’1’. The latter way is preferable if the application requires a defined point where the frequency changes.

Switching to Slow Down operation affects frequency sensitive peripherals like serial interfaces, timers, PWM, etc. If these units are to be operated in Slow Down mode their precalers or reload values must be adapted. Please note that the reduced CPU frequency decreases e.g. timer resolution and increases the step width e.g. for baudrate generation. The oscillator frequency in such a case should be chosen to accomodate the required resolutions and/or baudrates.

SYSCON2

System Control Register 2 **ESFR (F1D0$_H$/E8$_H$)** **Reset value: 00X0$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLK LOCK	CLKREL					CLKCON		SCS	RCS	PDCON		SYSRLS			
rh	rw					rw		rw	rw	rw		rwh			

Bit	Function
SYSRLS	**SYSCON Release Function (Unlock field)** Must be written in a defined way in order to execute the unlock sequence. See separate description
PDCON	**Power Down Control (during power down mode)** 00: RTC = On, Ports = On (default after reset). 01: RTC = On, Ports = Off. 10: RTC = Off, Ports = On. 11: RTC = Off, Ports = Off.
RCS	**RTC Clock Source (not affected by a reset)** 0: Main oscillator. 1: Reserved.
SCS	**SDD Clock Source (not affected by a reset)** 0: Main oscillator. 1: Reserved.
CLKCON	**Clock State Control** 00: Running on configured basic frequency. 01: Running on slow down frequency, PLL ON if implemented. 10: Running on slow down frequency, PLL OFF if implemented. 11: Reserved. Do not use this combination.
CLKREL	**Reload Counter Value for Slowdown Divider** (SDD factor = CLKREL+1)
CLKLOCK	**Clock Signal Status Bit** 0: Main oscillator is unstable **or** PLL is unlocked (if PLL is implemented). 1: Main oscillator is stable **and** PLL is locked (if PLL is implemented). If no PLL is implemented it is assumed to be always locked.

Note: SYSCON2 (except for bitfield SYSRLS, of course) is write protected after the execution of EINIT unless it is released via the unlock sequence.

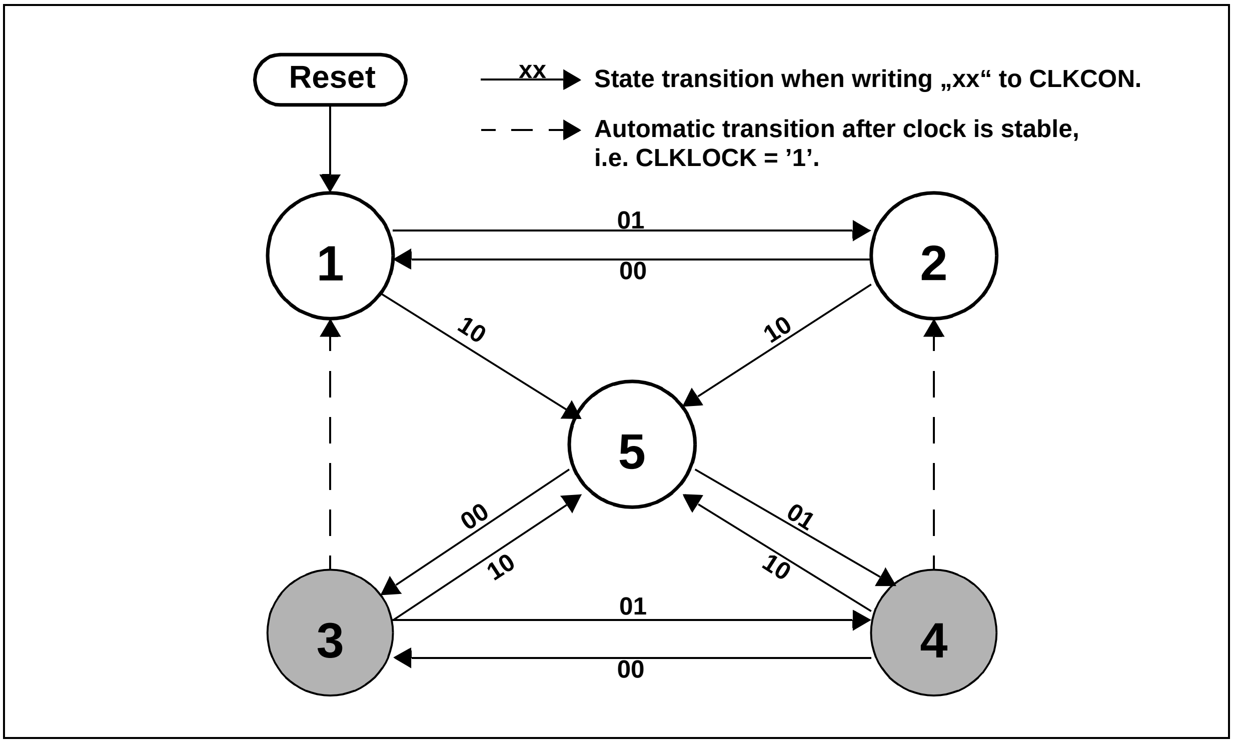

Reset
xx
State transition when writing „xx“ to CLKCON.
Automatic transition after clock is stable, i.e. CLKLOCK = ’1’.
1
2
3
4
5
01
00
10
10
00
10
01
10
01
00

Figure 21-4 Clock Switching State Machine

Table 21-3 Clock Switching State Description

State number	PLL status	f_{CPU} source	CLK CON	Note
1	Locked [1]	Basic	00	Standard operation on basic clock frequency.
2	Locked [1]	SDD	01	SDD operation with PLL On [1]. Fast (without delay) or manual switch back (from 5) to basic clock frequency.
3	Transient [1]	SDD	(00)	Intermediate state leading to state 1.
4	Transient [1]	SDD	(01)	Intermediate state leading to state 2.
5	Off	SDD	10	SDD operation with PLL Off. Reduced power consumption.

[1] The indicated PLL status only applies if the PLL is selected as the basic clock source. If the basic clock source is direct drive or prescaler the PLL will not lock.
If the oscillator watchdog is disabled (OWDDIS=’1’) the PLL will be off.

Note: When the PLL is the basic clock source and a reset occurs during SDD operation with the PLL off, the internal reset condition is extended so the PLL can lock before execution begins. The reset condition is terminated prematurely if no stable oscillator clock is detected. This ensures the operability of the device in case of a missing input clock signal.

21.5 Flexible Peripheral Management

The power consumed by the C164 also depends on the amount of active logic. Peripheral management enables the system designer to deactivate those on-chip peripherals that are not required in a given system status (e.g. a certain interface mode or standby). All modules that remain active, however, will still deliver their usual performance. If all modules that are fed by the peripheral clock driver (PCD) are disabled and also the other functions fed by the PCD are not required, this clock driver itself may also be disabled to save additional power.

This flexibility is realized by distributing the CPU clock via several clock drivers which can be separately controlled, and may also be smaller.

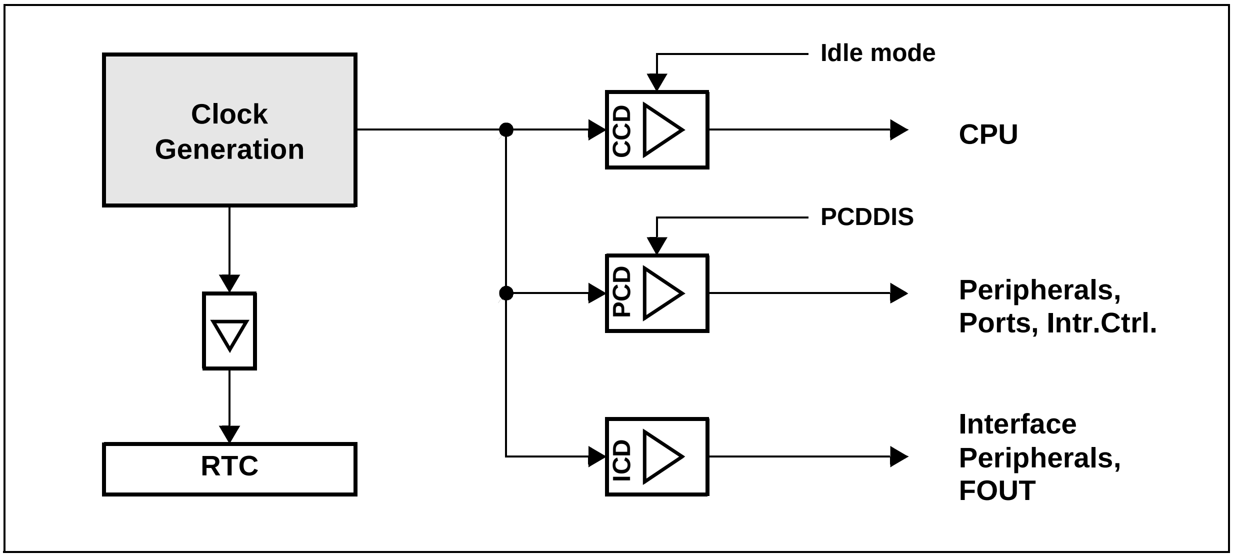


Figure 21-5 CPU Clock Distribution

Note: The Real Time Clock (RTC) is fed by a separate clock driver, so it can be kept running even in Power Down mode while still all the other circuitry is disconnected from the clock.

The registers of the generic peripherals can be accessed even while the respective module is disabled, as long as PCD is running (the registers of peripherals which are connected to ICD can be accessed even in this case, of course). The registers of X-peripherals cannot be accessed while the respective module is disabled by any means.

While a peripheral is disabled its output pins remain in the state they had at the time of disabling.

Software controls this flexible peripheral mangement via register SYSCON3 where each control bit is associated with an on-chip peripheral module.

SYSCON3
System Control Register 3 ESFR ($F1D4_H/EA_H$) Reset value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PCD DIS	-	CAN 1 DIS	-	-	-	-	CC6 DIS	CC2 DIS	-	PFM DIS	DFM DIS	GPT DIS	SSC DIS	ASC0 DIS	ADC DIS
rw	-	rw	-	-	-	-	rw	rw	-	rw	rw	rw	rw	rw	rw

Bit	**Function** (associated peripheral module)
ADCDIS	Analog/Digital Converter
ASC0DIS	USART ASC0
SSCDIS	Synchronous Serial Channel SSC
GPTDIS	General Purpose Timer Block GPT1
DFMDIS	On-chip Data Flash Memory Module [1]
PFMDIS	On-chip Program Flash Memory Module [1]
CC2DIS	CAPCOM2 Unit
CC6DIS	CAPCOM6 Unit
CAN1DIS	On-Chip CAN Module [2]
PCDDIS	Peripheral Clock Driver (also X-Peripherals)

[1] Bits PFMDIS and DFMDIS are only evaluated during Idle mode, i.e. the Flash modules are only switched off when entering Idle mode and are reactivated upon termination of Idle mode.

[2] When bit CANxDIS is cleared the CAN module is re-activated by an internal reset signal and must then be re-configured in order to operate properly.

Note: The allocation of peripheral disable bits within register SYSCON3 is device specific and may be different in other derivatives than the C164.
SYSCON3 is write protected after the execution of EINIT unless it is released via the unlock sequence.

When disabling the peripheral clock driver (PCD), the following details should be respected:

- The clock signal for all connected peripherals is stopped. Make sure that all peripherals enter a safe state before disabling PCD.
- The output signal CLKOUT will remain HIGH (FOUT will keep on toggling).
- Interrupt requests will still be recognized even while PCD is disabled.
- No new output values are gated from the port output latches to the output port pins and no new input values are latched from the input port pins.
- No register access is possible for generic peripherals
(register access is possible for individually disabled generic peripherals,
no register access at all is possible for disabled X-Peripherals)

21.6 Programmable Frequency Output Signal

The system clock output (CLKOUT) can be replaced by the programmable frequency output signal f_{OUT}. This signal can be controlled via software (contrary to CLKOUT), and so can be adapted to the requirements of the connected external circuitry. The programmability also extends the power management to a system level, as also circuitry (peripherals, etc.) outside the C164 can be influenced, i.e. run at a scalable frequency or temporarily can be switched off completely.

This clock signal is generated via a reload counter, so the output frequency can be selected in small steps. An optional toggle latch provides a clock signal with a 50% duty cycle.

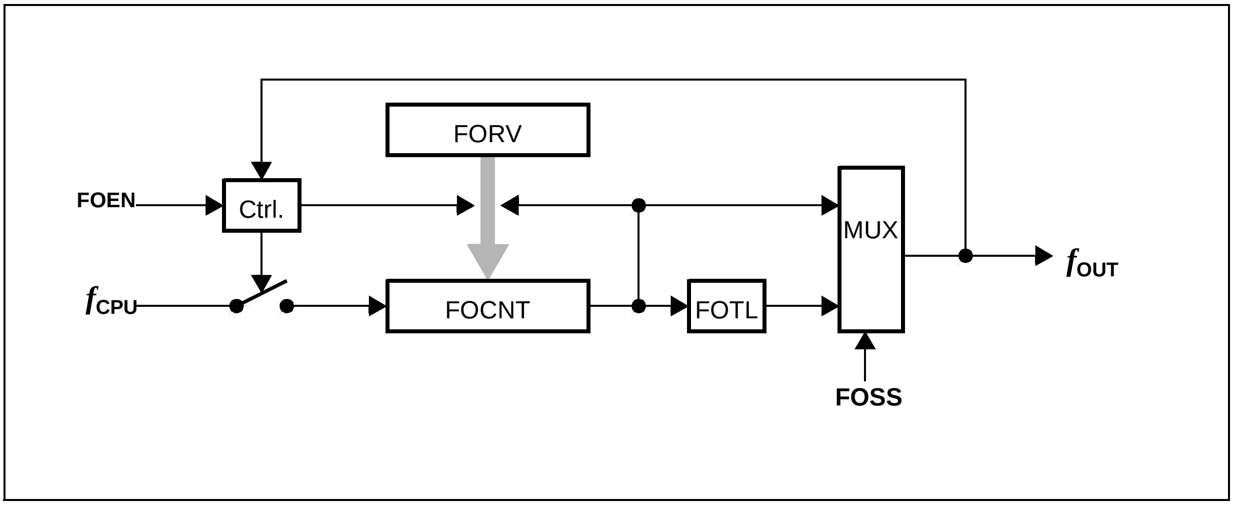


Figure 21-6 Clock Output Signal Generation

Signal f_{OUT} always provides complete output periods (see Signal Waveforms below):

- When f_{OUT} is started (FOEN-->'1') FOCNT is loaded from FORV
- When f_{OUT} is stopped (FOEN-->'0') FOCNT is stopped when f_{OUT} has reached (or is) '0'.

Signal f_{OUT} is independent from the peripheral clock driver PCD. While CLKOUT would stop when PCD is disabled, f_{OUT} will keep on toggling. Thus external circuitry may be controlled independent from on-chip peripherals.

Note: Counter FOCNT is clocked with the CPU clock signal f_{CPU} (see figure above) and therefore will also be influenced by the SDD operation.

Register FOCON provides control over the output signal generation (frequency, waveform, activation) as well as all status information (counter value, FOTL).

FOCON

Frequency Output Ctrl. Reg. SFR (FFAA$_H$/D5$_H$) Reset value: 0000$_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FOEN	FOSS	FORV						-	FOTL	FOCNT					
rw	rw	rw							rwh	rwh					

Bit	Function
FOCNT	**Frequency Output Counter**
FOTL	**Frequency Output Toggle Latch** Is toggled upon each underflow of FOCNT.
FORV	**Frequency Output Reload Value** Is copied to FOCNT upon each underflow of FOCNT.
FOSS	**Frequency Output Signal Select** 0: Output of the toggle latch: DC=50%. 1: Output of the reload counter: DC depends on FORV.
FOEN	**Frequency Output Enable** 0: Frequency output generation stops when signal f_{OUT} is/gets low. 1: FOCNT is running, f_{OUT} is gated to pin. 1st reload after 0-1 transition.

Note: *It is not recommended to write to any part of bitfield FOCNT, especially not while the counter is running. Writing to FOCNT prior to starting the counter is obsolete because it will immediatley be reloaded from FORV. Writing to FOCNT during operation may produce unintended counter values.*

Signal f_{OUT} in the C164 is an alternate output function and shares a port pin with signal CLKOUT.
A priority ranking determines which function controls the shared pin:

Table 21-4 Priority Ranking for Shared Output Pin

Priority	Function	Control
1	CLKOUT	CLKEN = ’1’, FOEN = ’x’
2	FOUT	CLKEN = ’0’, FOEN = ’1’
3	General purpose IO	CLKEN = ’0’, FOEN = ’0’

Note: *For the generation of f_{OUT} pin FOUT must be switched to output, i.e. DP3.15=’1’. While f_{OUT} is disabled the pin is controlled by the port latch (see figure above). The port latch P3.15 must be ’0’ in order to maintain the f_{OUT} inactive level on the pin.*

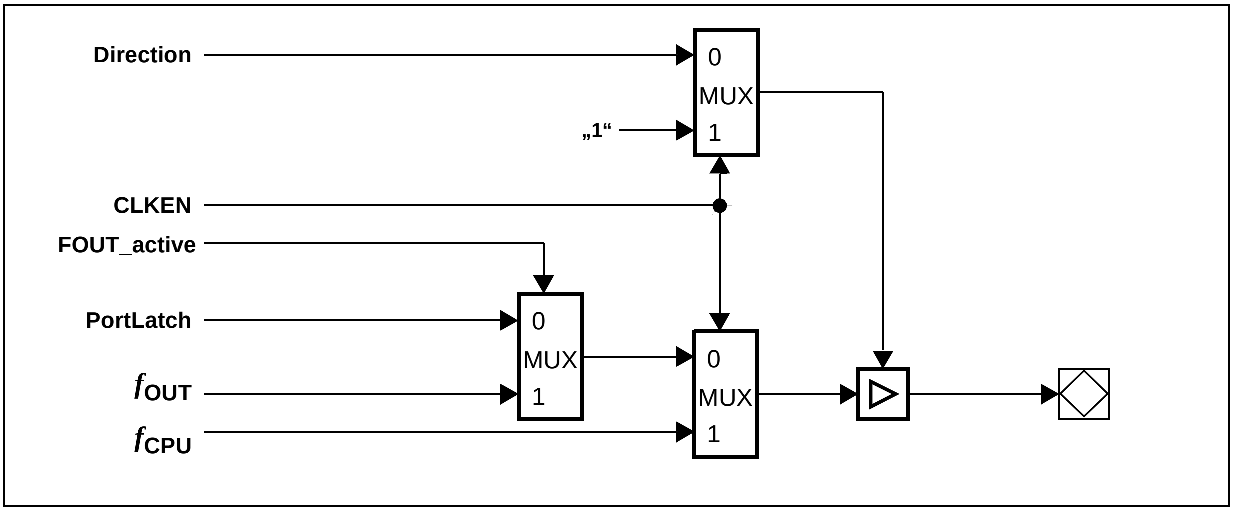


Figure 21-7 Connection to Port Logic (Functional Approach)

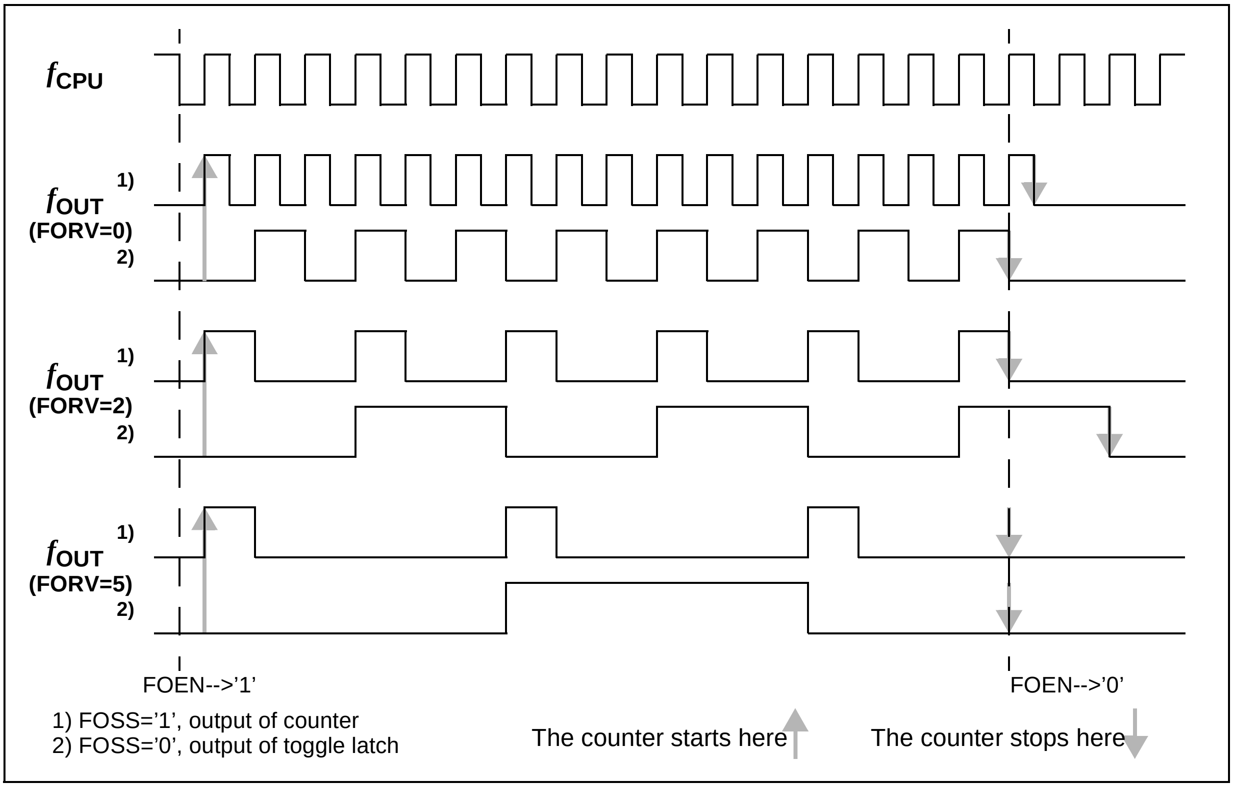


Figure 21-8 Signal Waveforms

Note: The output signal (for FOSS='1') is high for the duration of 1 f_{CPU} cycle for all reload values FORV > 0. For FORV = 0 the output signal corresponds to f_{CPU}.

Output Frequency Calculation

The output frequency can be calculated as $f_{OUT} = f_{CPU}$ / ((FORV+1) * $2^{(1\text{-}FOSS)}$),
so $f_{OUTmin} = f_{CPU}$ / 128 (FORV = $3F_H$, FOSS = ’0’),
and $f_{OUTmax} = f_{CPU}$ / 1 (FORV = 00_H, FOSS = ’1’).

Table 21-5 Selectable Output Frequency Range for f_{OUT}.

f_{CPU}	f_{OUT} in KHz for FORV=xx_H, FOSS=1/0					FORV for f_{OUT}=1 MHz	
	00_H	**01_H**	**02_H**	**$3E_H$**	**$3F_H$**	**FOSS=0**	**FOSS=1**
4 MHz	4000 2000	2000 1000	1333.33 666.667	63.492 31.746	62.5 31.25	01_H	03_H
10 MHz	10000 5000	5000 2500	3333.33 1666.667	158.73 79.365	156.25 78.125	04_H	09_H
12 MHz	12000 6000	6000 3000	4000 2000	190.476 95.238	187.5 93.75	05_H	$0B_H$
16 MHz	16000 8000	8000 4000	5333.33 2666.667	253.968 126.984	250 125	07_H	$0F_H$
20 MHz	20000 10000	10000 5000	6666.667 3333.33	317.46 158.73	312.5 156.25	09_H	13_H
25 MHz	25000 12500	12500 6250	8333.33 4166.667	396.825 198.4126	390.625 195.3125	$0C_H$ (1.04167)	17_H

21.7 Security Mechanism

The power management control registers (SYSCON1, SYSCON2, SYSCON3) control functions and modes which are critical for the C164's operation. For this reason they are locked (except for bitfield SYSRLS in register SYSCON2) after the execution of EINIT (like register SYSCON) so these vital system functions cannot be changed inadvertently e.g. by software errors. However, as these registers control the power management they need to be accessed during operation to select the appropriate mode. The system control software gets this access via a special unlock sequence which allows **one single** write access to either SYSCON1, SYSCON2, or SYSCON3 when executed properly. This provides a maximum of security.

Note: Of course SYSCON1, SYSCON2, and SYSCON3 may be read at any time without restrictions.

The unlock sequence is executed by writing defined values to bitfield SYSRLS using defined instructions (see table below). The instructions of the unlock sequence (including the intended write access) must be secured with an EXTR instruction (switch to ESFR space and lock interrupts).

Note: The unlock sequence is aborted if the locked range (EXTR) does not cover the complete sequence.
The unlock sequence provides no write access to register SYSCON.

Table 21-6 Definition of Unlock Sequence

Step	SYSRLS	Instruction	Notes
---	0000_B[1]	---	Status before release sequence
1	1001_B	BFLDL, OR, ORB[2], XOR, XORB[2]	Read-Modify-Write access
2	0011_B	MOV, MOVB[2], MOVBS[2], MOVBZ[2]	Write access
3	0111_B	BSET, BMOV[2], BMOVN[2], BOR[2], BXOR[1]	Read-Modify-Write access, bit instruction
4	---	---	Single (read-modify-)write access to SYSCON1, SYSCON2, or SYSCON3.
---	0000_B[3]	---	Status after release sequence

[1] SYSRLS must be set to 0000_B before the first step, if any OR command is used.

[2] Usually byte accesses should not be used for special function registers.

[3] SYSRLS is cleared by hardware if unlock sequence and write access were successful. SYSRLS shows the last value written otherwise.

The code examples below show how an access to SYSCON2/SYSCON3 can be accomplished in an application.

Examples where the PLL keeps running:

```
ENTER_SLOWDOWN:                   ;Currently running on basic clk. frequency
EXTR            #4H               ;Switch to ESFR space and lock sequence
BFLDL           SYSCON2,#0FH,#09H ;Unlock sequence, step 1 (1001B)
MOV             SYSCON2,#0003H    ;Unlock sequence, step 2 (0011B)
BSET            SYSCON2.2         ;Unlock sequence, step 3 (0111B)
                                  ;Single access to SYSCON2/SYSCON3
BFLDH           SYSCON2,#03H,#01H ;CLKCON=01B --> SDD frequency, PLL on
```

```
EXIT_SLOWDOWN:                    ;Currently running on SDD frequency
EXTR            #4H               ;Switch to ESFR space and lock sequence
BFLDL           SYSCON2,#0FH,#09H ;Unlock sequence, step 1 (1001B)
MOV             SYSCON2,#0003H    ;Unlock sequence, step 2 (0011B)
BSET            SYSCON2.2         ;Unlock sequence, step 3 (0111B)
                                  ;Single access to SYSCON2/SYSCON3
BFLDH           SYSCON2,#03H,#00H ;CLKCON=00B --> basic frequency
```

Examples where the PLL is disabled:

```
ENTER_SLOWDOWN:                   ;Currently running on basic clk. frequency
EXTR            #1H               ;Next access to ESFR space
BCLR            ISNC.2            ;PLLIE='0', i.e. PLL interrupt disabled
EXTR            #4H               ;Switch to ESFR space and lock sequence
BFLDL           SYSCON2,#0FH,#09H ;Unlock sequence, step 1 (1001B)
MOV             SYSCON2,#0003H    ;Unlock sequence, step 2 (0011B)
BSET            SYSCON2.2         ;Unlock sequence, step 3 (0111B)
                                  ;Single access to SYSCON2/SYSCON3
BFLDH           SYSCON2,#03H,#02H ;CLKCON=10B --> SDD frequency, PLL off


SDD_EXIT_AUTO:                    ;Currently running on SDD frequency
EXTR            #4H               ;Switch to ESFR space and lock sequence
BFLDL           SYSCON2,#0FH,#09H ;Unlock sequence, step 1 (1001B)
MOV             SYSCON2,#0003H    ;Unlock sequence, step 2 (0011B)
BSET            SYSCON2.2         ;Unlock sequence, step 3 (0111B)
                                  ;Single access to SYSCON2/SYSCON3
BFLDH           SYSCON2,#03H,#00H ;CLKCON=00B --> basic frequ./start PLL
EXTR            #1H               ;Next access to ESFR space
BSET            ISNC.2            ;PLLIE='1', i.e. PLL interrupt enabled


SDD_EXIT_MANUAL:                  ;Currently running on SDD frequency
EXTR            #4H               ;Switch to ESFR space and lock sequence
BFLDL           SYSCON2,#0FH,#09H ;Unlock sequence, step 1 (1001B)
MOV             SYSCON2,#0003H    ;Unlock sequence, step 2 (0011B)
BSET            SYSCON2.2         ;Unlock sequence, step 3 (0111B)
                                  ;Single access to SYSCON2/SYSCON3
BFLDH           SYSCON2,#03H,#01H ;CLKCON=01B --> stay on SDD/start PLL

USER_CODE:                        ;Space for any user code that...
                                  ;...must or can be executed before...
                                  ;...switching back to basic clock
CLOCK_OK:
EXTR            #1H               ;Next access to ESFR space
JNB             SYSCON2.15, CLOCK_OK;Wait until CLKLOCK='1'

EXTR            #4H               ;Switch to ESFR space and lock sequence
BFLDL           SYSCON2,#0FH,#09H ;Unlock sequence, step 1 (1001B)
MOV             SYSCON2,#0003H    ;Unlock sequence, step 2 (0011B)
BSET            SYSCON2.2         ;Unlock sequence, step 3 (0111B)
                                  ;Single access to SYSCON2/SYSCON3
BFLDH           SYSCON2,#03H,#00H ;CLKCON=00B --> basic frequency
EXTR            #1H               ;Next access to ESFR space
BSET            ISNC.2            ;PLLIE='1', i.e. PLL interrupt enabled
```

22 System Programming

To aid in software development, a number of features has been incorporated into the instruction set of the C164, including constructs for modularity, loops, and context switching. In many cases commonly used instruction sequences have been simplified while providing greater flexibility. The following programming features help to fully utilize this instruction set.

Instructions Provided as Subsets of Instructions

In many cases, instructions found in other microcontrollers are provided as subsets of more powerful instructions in the C164. This allows the same functionality to be provided while decreasing the hardware required and decreasing decode complexity. In order to aid assembly programming, these instructions, familiar from other microcontrollers, can be built in macros, thus providing the same names.

Directly Substitutable Instructions are instructions known from other microcontrollers that can be replaced by the following instructions of the C164:

Table 22-1 Substitution of Instructions

Substituted Instruction		C164 Instruction		Function
CLR	Rn	AND	Rn, $\#0_H$	Clear register
CPLB	Bit	BMOVN	Bit, Bit	Complement bit
DEC	Rn	SUB	Rn, $\#1_H$	Decrement register
INC	Rn	ADD	Rn, $\#1_H$	Increment register
SWAPB	Rn	ROR	Rn, $\#8_H$	Swap bytes within word

Modification of System Flags is performed using bit set or bit clear instructions (BSET, BCLR). All bit and word instructions can access the PSW register, so no instructions like CLEAR CARRY or ENABLE INTERRUPTS are required.

External Memory Data Access does not require special instructions to load data pointers or explicitly load and store external data. The C164 provides a Von-Neumann memory architecture and its on-chip hardware automatically detects accesses to internal RAM, GPRs, and SFRs.

Multiplication and Division

Multiplication and division of words and double words is provided through multiple cycle instructions implementing a Booth algorithm. Each instruction implicitly uses the 32-bit register MD (MDL = lower 16 bits, MDH = upper 16 bits). The MDRIU flag (Multiply or Divide Register In Use) in register MDC is set whenever either half of this register is written to or when a multiply/divide instruction is started. It is cleared whenever the MDL register is read. Because an interrupt can be acknowledged before the contents of register MD are saved, this flag is required to alert interrupt routines, which require the use of the multiply/divide hardware, so they can preserve register MD. This register, however, only needs to be saved when an interrupt routine requires use of the MD register and a previous task has not saved the current result. This flag is easily tested by the Jump-on-Bit instructions.

Multiplication or division is simply performed by specifying the correct (signed or unsigned) version of the multiply or divide instruction. The result is then stored in register MD. The overflow flag (V) is set if the result from a multiply or divide instruction is greater than 16 bits. This flag can be used to determine whether both word halfs must be transferred from register MD. The high portion of register MD (MDH) must be moved into the register file or memory first, in order to ensure that the MDRIU flag reflects the correct state.

The following instruction sequence performs an unsigned 16 by 16-bit multiplication:

```
SAVE:
JNB         MDRIU, START      ;Test if MD was in use.
SCXT        MDC, #0010H       ;Save and clear control register,
                              ;leaving MDRIU set
                              ;(only required for interrupted
                              ;multiply/divide instructions)
BSET        SAVED             ;Indicate the save operation
PUSH        MDH               ;Save previous MD contents...
PUSH        MDL               ;...on system stack
START:
MULU        R1, R2            ;Multiply 16·16 unsigned, Sets MDRIU
JMPR        cc_NV, COPYL      ;Test for only 16-bit result
MOV         R3, MDH           ;Move high portion of MD
COPYL:
MOV         R4, MDL           ;Move low portion of MD, Clears MDRIU
RESTORE:
JNB         SAVED, DONE       ;Test if MD registers were saved
POP         MDL               ;Restore registers
POP         MDH
POP         MDC
BCLR        SAVED             ;Multiplication is completed,
                              ;program continues
DONE:       ...
```

The above save sequence and the restore sequence after COPYL are only required if the current routine could have interrupted a previous routine which contained a MUL or DIV instruction. Register MDC is also saved because it is possible that a previous routine's Multiply or Divide instruction was interrupted while in progress. In this case the information about how to restart the instruction is contained in this register. Register MDC must be cleared to be correctly initialized for a subsequent multiplication or division. The old MDC contents must be popped from the stack before the RETI instruction is executed.

For a division the user must first move the dividend into the MD register. If a 16/16-bit division is specified, only the low portion of register MD must be loaded. The result is also stored into register MD. The low portion (MDL) contains the integer result of the division, while the high portion (MDH) contains the remainder.

The following instruction sequence performs a 32 by 16-bit division:

```
MOV         MDH, R1           ;Move dividend to MD register. Sets MDRIU
MOV         MDL, R2           ;Move low portion to MD
DIV         R3                ;Divide 32/16 signed, R3 holds divisor
JMPR        cc_V, ERROR       ;Test for divide overflow
MOV         R3, MDH           ;Move remainder to R3
MOV         R4, MDL           ;Move integer result to R4. Clears MDRIU
```

Whenever a multiply or divide instruction is interrupted while in progress, the address of the interrupted instruction is pushed onto the stack and the MULIP flag in the PSW of the interrupting routine is set. When the interrupt routine is exited with the RETI instruction, this bit is implicitly tested before the old PSW is popped from the stack. If MULIP='1' the multiply/divide instruction is re-read from the location popped from the stack (return address) and will be completed after the RETI instruction has been executed.

*Note: The MULIP flag is part of the **context of the interrupted task**. When the interrupting routine does not return to the interrupted task (e.g. scheduler switches to another task) the MULIP flag must be set or cleared according to the context of the task that is switched to.*

BCD Calculations

No direct support for BCD calculations is provided in the C164. BCD calculations are performed by converting BCD data to binary data, performing the desired calculations using standard data types, and converting the result back to BCD data. Due to the enhanced performance of division instructions binary data is quickly converted to BCD data through division by 10_D. Conversion from BCD data to binary data is enhanced by multiple bit shift instructions. This provides similar performance compared to instructions directly supporting BCD data types, while no additional hardware is required.

22.1 Stack Operations

The C164 supports two types of stacks. The system stack is used implicitly by the controller and is located in the internal RAM. The user stack provides stack access to the user in either the internal or external memory. Both stack types grow from high memory addresses to low memory addresses.

Internal System Stack

A system stack is provided to store return vectors, segment pointers, and processor status for procedures and interrupt routines. A system register, SP, points to the top of the stack. This pointer is decremented when data is pushed onto the stack, and incremented when data is popped.

The internal system stack can also be used to temporarily store data or pass it between subroutines or tasks. Instructions are provided to push or pop registers on/from the system stack. However, in most cases the register banking scheme provides the best performance for passing data between multiple tasks.

Note: The system stack allows the storage of words only. Bytes must either be converted to words or the respective other byte must be disregarded.
Register SP can only be loaded with even byte addresses (The LSB of SP is always '0').

Detection of stack overflow/underflow is supported by two registers, STKOV (Stack Overflow Pointer) and STKUN (Stack Underflow Pointer). Specific system traps (Stack Overflow trap, Stack Underflow trap) will be entered whenever the SP reaches either boundary specified in these registers.

The contents of the stack pointer are compared to the contents of the overflow register, whenever the SP is DECREMENTED either by a CALL, PUSH or SUB instruction. An overflow trap will be entered, when the SP value is less than the value in the stack overflow register.

The contents of the stack pointer are compared to the contents of the underflow register, whenever the SP is INCREMENTED either by a RET, POP or ADD instruction. An underflow trap will be entered, when the SP value is greater than the value in the stack underflow register.

Note: When a value is MOVED into the stack pointer, NO check against the overflow/underflow registers is performed.

In many cases the user will place a software reset instruction (SRST) into the stack underflow and overflow trap service routines. This is an easy approach, which does not require special programming. However, this approach assumes that the defined internal stack is sufficient for the current software and that exceeding its upper or lower boundary represents a fatal error.

It is also possible to use the stack underflow and stack overflow traps to cache portions of a larger external stack. Only the portion of the system stack currently being used is placed into the internal memory, thus allowing a greater portion of the internal RAM to be used for program, data or register banking. This approach assumes no error but requires a set of control routines (see below).

Circular (virtual) Stack

This basic technique allows pushing until the overflow boundary of the internal stack is reached. At this point a portion of the stacked data must be saved into external memory to create space for further stack pushes. This is called “stack flushing”. When executing a number of return or pop instructions, the upper boundary (since the stack empties upward to higher memory locations) is reached. The entries that have been previously saved in external memory must now be restored. This is called “stack filling”. Because procedure call instructions do not continue to nest infinitely and call and return instructions alternate, flushing and filling normally occurs very infrequently. If this is not true for a given program environment, this technique should not be used because of the overhead of flushing and filling.

The basic mechanism is the transformation of the addresses of a virtual stack area, controlled via registers SP, STKOV and STKUN, to a defined physical stack area within the internal RAM via hardware. This virtual stack area covers all possible locations that SP can point to, i.e. 00’F000$_H$ through 00’FFFE$_H$. STKOV and STKUN accept the same 4 KByte address range.

The size of the physical stack area within the internal RAM that effectively is used for standard stack operations is defined via bitfield STKSZ in register SYSCON (see below).

Table 22-2 Circular Stack Address Transformation

STKSZ	Stack Size (Words)	Internal RAM Addresses (Words) of Physical Stack	Significant Bits of Stack Ptr. SP
0 0 0 $_B$	256	00’FBFE$_H$...00’FA00$_H$ (Default after Reset)	SP.8...SP.0
0 0 1 $_B$	128	00’FBFE$_H$...00’FB00$_H$	SP.7...SP.0
0 1 0 $_B$	64	00’FBFE$_H$...00’FB80$_H$	SP.6...SP.0
0 1 1 $_B$	32	00’FBFE$_H$...00’FBC0$_H$	SP.5...SP.0
1 0 0 $_B$	512	00’FBFE$_H$...00’F800$_H$ (not for 1KByte IRAM)	SP.9...SP.0
1 0 1 $_B$	---	Reserved. Do not use this combination.	---
1 1 0 $_B$	---	Reserved. Do not use this combination.	---
1 1 1 $_B$	1024	00’FDFE$_H$...00’FX00$_H$ (Note: No circular stack) 00’FX00$_H$ represents the lower IRAM limit, i.e. 1 KB: 00’FA00$_H$, 2 KB: 00’F600$_H$, 3 KB: 00’F200$_H$	SP.11...SP.0

The virtual stack addresses are transformed to physical stack addresses by concatenating the significant bits of the stack pointer register SP (see table) with the complementary most significant bits of the upper limit of the physical stack area (00’FBFE$_H$). This transformation is done via hardware (see figure below).

The reset values (STKOV=FA00$_H$, STKUN=FC00$_H$, SP=FC00$_H$, STKSZ=000$_B$) map the virtual stack area directly to the physical stack area and allow using the internal system stack without any changes, provided that the 256 word area is not exceeded.

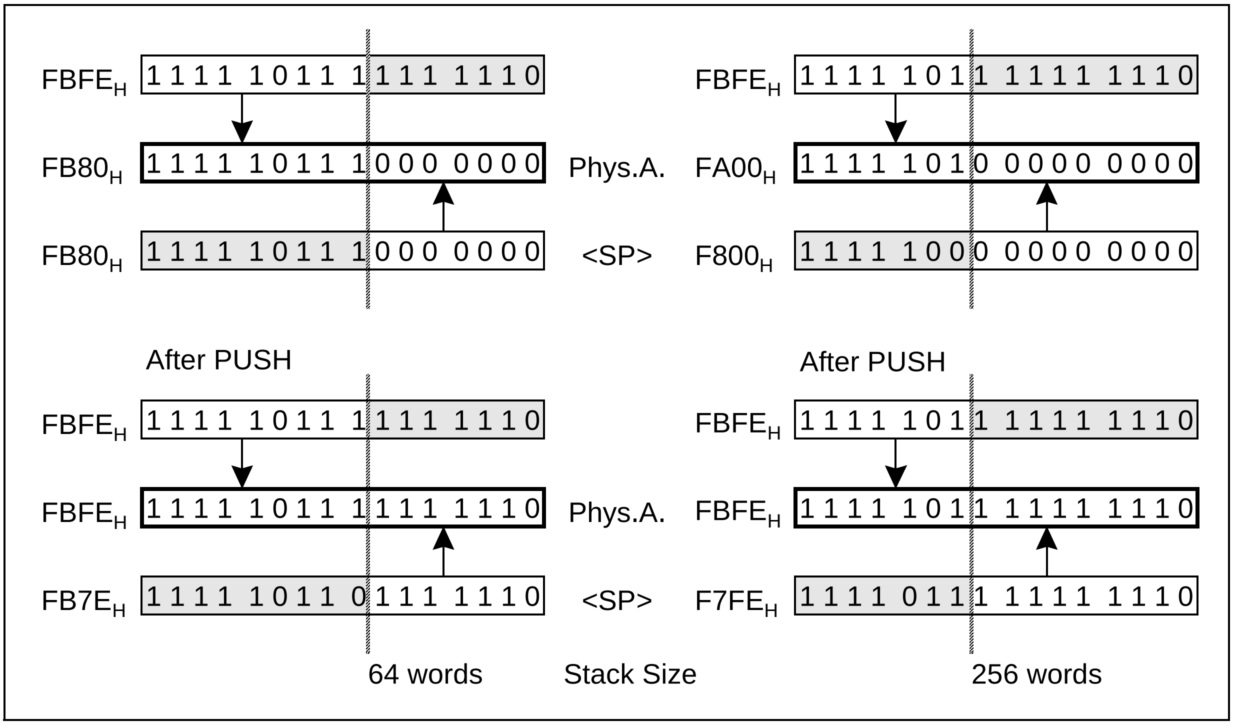


Figure 22-1 Physical Stack Address Generation

The following example demonstrates the circular stack mechanism which is also an effect of this virtual stack mapping: First, register R1 is pushed onto the lowest physical stack location according to the selected maximum stack size. With the following instruction, register R2 will be pushed onto the highest physical stack location although the SP is decremented by 2 as for the previous push operation.

```
MOV         SP, #0F802H       ;Set SP before last entry...
                              ;...of physical stack of 256 words
...                           ;(SP)=F802H: Physical stack addr.=FA02H
PUSH        R1                ;(SP)=F800H: Physical stack addr.=FA00H
PUSH        R2                ;(SP)=F7FEH: Physical stack addr.=FBFEH
```

The effect of the address transformation is that the physical stack addresses wrap around from the end of the defined area to its beginning. When flushing and filling the internal stack, this circular stack mechanism only requires to move that portion of stack data which is really to be re-used (i.e. the upper part of the defined stack area) instead of the whole stack area. Stack data that remain in the lower part of the internal stack need not be moved by the distance of the space being flushed or filled, as the stack pointer automatically wraps around to the beginning of the freed part of the stack area.

Note: This circular stack technique is applicable for stack sizes of 32 to 512 words (STKSZ = ‘000$_B$’ to ‘100$_B$’), it does not work with option STKSZ = ‘111$_B$’, which uses the complete internal RAM for system stack.
In the latter case the address transformation mechanism is deactivated.

When a boundary is reached, the stack underflow or overflow trap is entered, where the user moves a predetermined portion of the internal stack to or from the external stack. The amount of data transferred is determined by the average stack space required by routines and the frequency of calls, traps, interrupts and returns. In most cases this will be approximately one quarter to one tenth the size of the internal stack. Once the transfer is complete, the boundary pointers are updated to reflect the newly allocated space on the internal stack. Thus, the user is free to write code without concern for the internal stack limits. Only the execution time required by the trap routines affects user programs.

The following procedure initializes the controller for usage of the circular stack mechanism:

- Specify the size of the physical system stack area within the internal RAM (bitfield STKSZ in register SYSCON).
- Define two pointers, which specify the upper and lower boundary of the external stack. These values are then tested in the stack underflow and overflow trap routines when moving data.
- Set the stack overflow pointer (STKOV) to the limit of the defined internal stack area plus six words (for the reserved space to store two interrupt entries).

The internal stack will now fill until the overflow pointer is reached. After entry into the overflow trap procedure, the top of the stack will be copied to the external memory. The internal pointers will then be modified to reflect the newly allocated space. After exiting from the trap procedure, the internal stack will wrap around to the top of the internal stack, and continue to grow until the new value of the stack overflow pointer is reached.

When the underflow pointer is reached while the stack is meptied the bottom of stack is reloaded from the external memory and the internal pointers are adjusted accordingly.

Linear Stack

The C164 also offers a linear stack option (STKSZ = ‘111_B’), where the system stack may use the complete internal RAM area. This provides a large system stack without requiring procedures to handle data transfers for a circular stack. However, this method also leaves less RAM space for variables or code. The RAM area that may effectively be consumed by the system stack is defined via the STKUN and STKOV pointers. The underflow and overflow traps in this case serve for fatal error detection only.

For the linear stack option all modifiable bits of register SP are used to access the physical stack. Although the stack pointer may cover addresses from 00’$F000_H$ up to 00’$FFFE_H$ the (physical) system stack must be located within the internal RAM and therefore may only use the address range 00’$F600_H$ to 00’$FDFE_H$. It is the user’s responsibility to restrict the system stack to the internal RAM range.

Note: Avoid stack accesses below the IRAM area (ESFR space and reserved area) and within address range 00’$FE00_H$ and 00’$FFFE_H$ (SFR space).
Otherwise unpredictable results will occur.

User Stacks

User stacks provide the ability to create task specific data stacks and to off-load data from the system stack. The user may push both bytes and words onto a user stack, but is responsible for using the appropriate instructions when popping data from the specific user stack. No hardware detection of overflow or underflow of a user stack is provided. The following addressing modes allow implementation of user stacks:

[– Rw], Rb or [– Rw], Rw: Pre-decrement Indirect Addressing.
Used to push one byte or word onto a user stack. This mode is only available for MOV instructions and can specify any GPR as the user stack pointer.

Rb, [Rw_i+] or Rw, [Rw_i+]: Post-increment Index Register Indirect Addressing.
Used to pop one byte or word from a user stack. This mode is available to most instructions, but only GPRs R0-R3 can be specified as the user stack pointer.

Rb, [Rw+] or Rw, [Rw+]: Post-increment Indirect Addressing.
Used to pop one byte or word from a user stack. This mode is only available for MOV instructions and can specify any GPR as the user stack pointer.

22.2 Register Banking

Register banking provides the user with an extremely fast method to switch user context. A single machine cycle instruction saves the old bank and enters a new register bank. Each register bank may assign up to 16 registers. Each register bank should be allocated during coding based on the needs of each task. Once the internal memory has been partitioned into a register bank space, internal stack space and a global internal memory area, each bank pointer is then assigned. Thus, upon entry into a new task, the appropriate bank pointer is used as the operand for the SCXT (switch context) instruction. Upon exit from a task a simple POP instruction to the context pointer (CP) restores the previous task's register bank.

22.3 Procedure Call Entry and Exit

To support modular programming a procedure mechanism is provided to allow coding of frequently used portions of code into subroutines. The CALL and RET instructions store and restore the value of the instruction pointer (IP) on the system stack before and after a subroutine is executed.

Procedures may be called conditionally with instructions CALLA or CALLI, or be called unconditionally using instructions CALLR or CALLS.

Note: Any data pushed onto the system stack during execution of the subroutine must be popped before the RET instruction is executed.

Passing Parameters on the System Stack

Parameters may be passed via the system stack through PUSH instructions before the subroutine is called, and POP instructions during execution of the subroutine. Base plus offset indirect addressing also permits access to parameters without popping these parameters from the stack during execution of the subroutine. Indirect addressing provides a mechanism of accessing data referenced by data pointers, which are passed to the subroutine.

In addition, two instructions have been implemented to allow one parameter to be passed on the system stack without additional software overhead.

The PCALL (push and call) instruction first pushes the 'reg' operand and the IP contents onto the system stack and then passes control to the subroutine specified by the 'caddr' operand.

When exiting from the subroutine, the RETP (return and pop) instruction first pops the IP and then the 'reg' operand from the system stack and returns to the calling program.

Cross Segment Subroutine Calls

Calls to subroutines in different segments require the use of the CALLS (call inter-segment subroutine) instruction. This instruction preserves both the CSP (code segment pointer) and IP on the system stack.

Upon return from the subroutine, a RETS (return from inter-segment subroutine) instruction must be used to restore both the CSP and IP. This ensures that the next instruction after the CALLS instruction is fetched from the correct segment.

Note: It is possible to use CALLS within the same segment, but still two words of the stack are used to store both the IP and CSP.

Providing Local Registers for Subroutines

For subroutines which require local storage, the following methods are provided:

Alternate Bank of Registers: Upon entry into a subroutine, it is possible to specify a new set of local registers by executing the SCXT (switch context) instruction. This mechanism does not provide a method to recursively call a subroutine.

Saving and Restoring of Registers: To provide local registers, the contents of the registers which are required for use by the subroutine can be pushed onto the stack and the previous values be popped before returning to the calling routine. This is the most common technique used today and it does provide a mechanism to support recursive procedures. This method, however, requires two machine cycles per register stored on the system stack (one cycle to PUSH the register, and one to POP the register).

Use of the System Stack for Local Registers: It is possible to use the SP and CP to set up local subroutine register frames. This enables subroutines to dynamically allocate local variables as needed within two machine cycles. A local frame is allocated by simply subtracting the number of required local registers from the SP, and then moving the value of the new SP to the CP.

This operation is supported through the SCXT (switch context) instruction with the addressing mode ’reg, mem’. Using this instruction saves the old contents of the CP on the system stack and moves the value of the SP into CP (see example below). Each local register is then accessed as if it was a normal register. Upon exit from the subroutine, first the old CP must be restored by popping it from the stack and then the number of used local registers must be added to the SP to restore the allocated local space back to the system stack.

Note: The system stack is growing downwards, while the register bank is growing upwards.

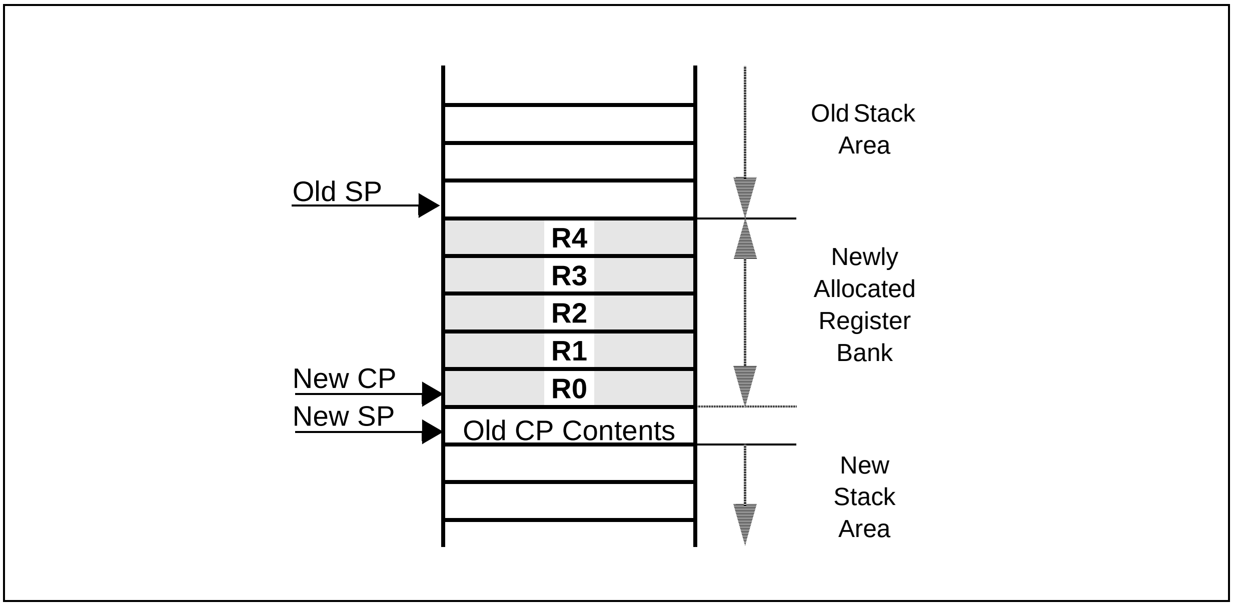


Figure 22-2 Local Registers

The software to provide the local register bank for the example above is very compact:

After entering the subroutine:

```
SUB         SP, #10D          ;Free 5 words in the current system stack
SCXT        CP, SP            ;Set the new register bank pointer
```

Before exiting the subroutine:

```
POP         CP                ;Restore the old register bank
ADD         SP, #10D          ;Release the 5 words...
                              ;...of the current system stack
```

22.4 Table Searching

A number of features have been included to decrease the execution time required to search tables. First, branch delays are eliminated by the branch target cache after the first iteration of the loop. Second, in non-sequentially searched tables, the enhanced performance of the ALU allows more complicated hash algorithms to be processed to obtain better table distribution. For sequentially searched tables, the auto-increment indirect addressing mode and the E (end of table) flag stored in the PSW decrease the number of overhead instructions executed in the loop.

The two examples below illustrate searching ordered tables and non-ordered tables, respectively:

```
MOV         R0, #BASE         ;Move table base into R0
LOOP:
CMP         R1, [R0+]         ;Compare target to table entry
JMPR        cc_SGT, LOOP      ;Test whether target has not been found
```

Note: The last entry in the table must be greater than the largest possible target.

```
MOV         R0, #BASE         ;Move table base into R0
LOOP:
CMP         R1, [R0+]         ;Compare target to table entry
JMPR        cc_NET, LOOP      ;Test whether target is not found AND..
                              ;..the end of table has not been reached.
```

Note: The last entry in the table must be equal to the lowest signed integer (8000_H).

22.5 Floating Point Support

All floating point operations are performed using software. Standard multiple precision instructions are used to perform calculations on data types that exceed the size of the ALU. Multiple bit rotate and logic instructions allow easy masking and extracting of portions of floating point numbers.

To decrease the time required to perform floating point operations, two hardware features have been implemented in the CPU core. First, the PRIOR instruction aids in normalizing floating point numbers by indicating the position of the first set bit in a GPR. This result can the be used to rotate the floating point result accordingly. The second feature aids in properly rounding the result of normalized floating point numbers through the overflow (V) flag in the PSW. This flag is set when a one is shifted out of the carry bit during shift right operations. The overflow flag and the carry flag are then used to round the floating point result based on the desired rounding algorithm.

22.6 Peripheral Control and Interface

All communication between peripherals and the CPU is performed either by PEC transfers to and from internal memory, or by explicitly addressing the SFRs associated with the specific peripherals. After resetting the C164 all peripherals (except the watchdog timer) are disabled and initialized to default values. A desired configuration of a specific peripheral is programmed using MOV instructions of either constants or memory values to specific SFRs. Specific control flags may also be altered via bit instructions.

Once in operation, the peripheral operates autonomously until an end condition is reached at which time it requests a PEC transfer or requests CPU servicing through an interrupt routine. Information may also be polled from peripherals through read accesses to SFRs or bit operations including branch tests on specific control bits in SFRs. To ensure proper allocation of peripherals among multiple tasks, a portion of the internal memory has been made bit addressable to allow user semaphores. Instructions have also been provided to lock out tasks via software by setting or clearing user specific bits and conditionally branching based on these specific bits.

It is recommended that bit fields in control SFRs are updated using the BFLDH and BFLDL instructions or a MOV instruction to avoid undesired intermediate modes of operation which can occur, when BCLR/BSET or AND/OR instruction sequences are used.

22.7 Trap/Interrupt Entry and Exit

Interrupt routines are entered when a requesting interrupt has a priority higher than the current CPU priority level. Traps are entered regardless of the current CPU priority. When either a trap or interrupt routine is entered, the state of the machine is preserved on the system stack and a branch to the appropriate trap/interrupt vector is made.

All trap and interrupt routines require the use of the RETI (return from interrupt) instruction to exit from the called routine. This instruction restores the system state from the system stack and then branches back to the location where the trap or interrupt occurred.

22.8 Unseparable Instruction Sequences

The instructions of the C164 are very efficient (most instructions execute in one machine cycle) and even the multiplication and division are interruptable in order to minimize the response latency to interrupt requests (internal and external). In many microcontroller applications this is vital.

Some special occasions, however, require certain code sequences (e.g. semaphore handling) to be uninterruptable to function properly. This can be provided by inhibiting interrupts during the respective code sequence by disabling and enabling them before and after the sequence. The necessary overhead may be reduced by means of the ATOMIC instruction which allows locking 1...4 instructions to an unseparable code sequence, during which the interrupt system (standard interrupts and PEC requests) **and Class A Traps** ($\overline{\text{NMI}}$, stack overflow/underflow) are disabled. A **Class B Trap** (illegal opcode, illegal bus access, etc.), however, will interrupt the atomic sequence, since it indicates a severe hardware problem.

The interrupt inhibit caused by an ATOMIC instruction gets active immediately, i.e. no other instruction will enter the pipeline except the one that follows the ATOMIC instruction, and no interrupt request will be serviced in between. All instructions requiring multiple cycles or hold states are regarded as one instruction in this sense (e.g. MUL is one instruction). Any instruction type can be used within an unseparable code sequence.

```
ATOMIC      #3                  ;The next 3 instr. are locked (No NOP requ.)
MOV         R0, #1234H          ;Instr. 1 (no other instr. enters pipeline!)
MOV         R1, #5678H          ;Instr. 2
MUL         R0, R1              ;Instr. 3: MUL regarded as one instruction
MOV         R2, MDL             ;This instruction is out of the scope...
                                ;...of the ATOMIC instruction sequence
```

22.9 Overriding the DPP Addressing Mechanism

The standard mechanism to access data locations uses one of the four data page pointers (DPPx), which selects a 16 KByte data page, and a 14-bit offset within this data page. The four DPPs allow immediate access to up to 64 KByte of data. In applications with big data arrays, especially in HLL applications using large memory models, this may require frequent reloading of the DPPs, even for single accesses.

The EXTP (extend page) instruction allows switching to an arbitrary data page for 1...4 instructions without having to change the current DPPs.

```
EXTP        R15, #1             ;The override page number is stored in R15
MOV         R0, [R14]           ;The (14-bit) page offset is stored in R14
MOV         R1, [R13]           ;This instruction uses the std. DPP scheme!
```

The EXTS (extend segment) instruction allows switching to a 64 KByte segment oriented data access scheme for 1...4 instructions without having to change the current DPPs. In this case all 16 bits of the operand address are used as segment offset, with the segment taken from the EXTS instruction. This greatly simplifies address calculation with continuous data like huge arrays in "C".

```
EXTS        #15, #1          ;The override seg. is 15 (0F'0000H..0F'FFFFH)
MOV         R0, [R14]        ;The (16-bit) segment offset is stored in R14
MOV         R1, [R13]        ;This instruction uses the std. DPP scheme!
```

Note: Instructions EXTP and EXTS inhibit interrupts the same way as ATOMIC.

Short Addressing in the Extended SFR (ESFR) Space

The short addressing modes of the C164 (REG or BITOFF) implicitly access the SFR space. The additional ESFR space would have to be accessed via long addressing modes (MEM or [Rw]). The EXTR (extend register) instruction redirects accesses in short addressing modes to the ESFR space for 1...4 instructions, so the additional registers can be accessed this way, too.

The EXTPR and EXTSR instructions combine the DPP override mechanism with the redirection to the ESFR space using a single instruction.

Note: Instructions EXTR, EXTPR and EXTSR inhibit interrupts the same way as ATOMIC. The switching to the ESFR area and data page overriding is checked by the development tools or handled automatically.

Nested Locked Sequences

Each of the described extension instruction and the ATOMIC instruction starts an internal "extension counter" counting the effected instructions. When another extension or ATOMIC instruction is contained in the current locked sequence this counter is restarted with the value of the new instruction. This allows the construction of locked sequences longer than 4 instructions.

Note: • *Interrupt latencies may be increased when using locked code sequences.*
• *PEC requests are not serviced during idle mode, if the IDLE instruction is part of a locked sequence.*

22.10 Handling the Internal Code Memory

The Mask-ROM/OTP/Flash versions of the C164 provide on-chip code memory that may store code as well as data. The lower 32 KByte of this code memory are referred to as the „internal ROM area“. Access to this internal ROM area is controlled during the reset configuration and via software. The ROM area may be mapped to segment 0, to segment 1 or the code memory may be disabled at all.

Note: The internal ROM area always occupies an address area of 32 KByte, even if the implemented mask ROM/OTP/Flash memory is smaller than that (e.g. 8 KByte). Of course the total implemented memory may exceed 32 KBytes.

Code Memory Configuration during Reset

The control input pin $\overline{\text{EA}}$ (External Access) enables the user to define the address area from which the first instructions after reset are fetched. When $\overline{\text{EA}}$ is low (‘0’) during reset, the internal code memory is disabled and the first instructions are fetched from external memory. When $\overline{\text{EA}}$ is high (‘1’) during reset, the internal code memory is globally enabled and the first instructions are fetched from the internal memory.

Note: Be sure not to select internal memory access after reset on ROMless devices.

Mapping the Internal ROM Area

After reset the internal ROM area is mapped into segment 0, the “system segment” (00’0000$_H$...00’7FFF$_H$) as a default. This is necessary to allow the first instructions to be fetched from locations 00’0000$_H$ ff. The ROM area may be mapped to segment 1 (01’0000$_H$...01’7FFF$_H$) by setting bit ROMS1 in register SYSCON. The internal ROM area may now be accessed through the lower half of segment 1, while accesses to segment 0 will now be made to external memory. This adds flexibility to the system software. The interrupt/trap vector table, which uses locations 00’0000$_H$ through 00’01FF$_H$, is now part of the external memory and may therefore be modified, i.e. the system software may now change interrupt/trap handlers according to the current condition of the system. The internal code memory can still be used for fixed software routines like IO drivers, math libraries, application specific invariant routines, tables, etc. This combines the advantage of an integrated non-volatile memory with the advantage of a flexible, adaptable software system.

Enabling and Disabling the Internal Code Memory After Reset

If the internal code memory does not contain an appropriate startup code, the system may be booted from external memory, while the internal memory is enabled afterwards to provide access to library routines, tables, etc.

If the internal code memory only contains the startup code and/or test software, the system may be booted from internal memory, which may then be disabled, after the software has switched to executing from (e.g.) external memory, in order to free the address space occupied by the internal code memory, which is now unnecessary.

22.11 Pits, Traps and Mines

Although handling the internal code memory provides powerful means to enhance the overall performance and flexibility of a system, extreme care must be taken in order to avoid a system crash. Instruction memory is the most crucial resource for the C164 and it must be made sure that it never runs out of it. The following precautions help to take advantage of the methods mentioned above without jeopardizing system security.

Internal code memory access after reset: When the first instructions are to be fetched from internal memory (EA=‘1’), the device must contain code memory, and this must contain a valid reset vector and valid code at its destination.

Mapping the internal ROM area to segment 1: Due to instruction pipelining, any new ROM mapping will at the earliest become valid for the second instruction after the instruction which has changed the ROM mapping. To enable accesses to the ROM area after mapping a branch to the newly selected ROM area (JMPS) and reloading of all data page pointers is required.
This also applies to re-mapping the internal ROM area to segment 0.

Enabling the internal code memory after reset: When enabling the internal code memory after having booted the system from external memory, note that the C164 will then access the internal memory using the current segment offset, rather than accessing external memory.

Disabling the internal code memory after reset: When disabling the internal code memory after having booted the system from there, note that the C164 will not access external memory before a jump to segment 0 (in this case) is executed.

General Rules

When mapping the code memory no instruction or data accesses should be made to the internal memory, otherwise unpredictable results may occur.

To avoid these problems, the instructions that configure the internal code memory should be executed from external memory or from the on-chip RAM.

Whenever the internal code memory is disabled, enabled or remapped the DPPs must be explicitly (re)loaded to enable correct data accesses to the internal and/or external memory.

23 The Register Set

This section summarizes all registers, which are implemented in the C164 and explains the description format which is used in the chapters describing the function and layout of the SFRs.
For easy reference the registers are ordered according to two different keys (except for GPRs):

- Ordered by address, to check which register a given address references,
- Ordered by register name, to find the location of a specific register.

23.1 Register Description Format

In the respective chapters the function and the layout of the SFRs is described in a specific format which provides a number of details about the described special function register. The example below shows how to interpret these details.

REG_NAME
Name of Register **E/SFR *(A16$_H$/A8$_H$)*** **Reset value: * * * *$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<empty for byte registers>								**std**	**hw**	**read/ write bit**	**read bit**	**write bit**	**bitfield**		
-	-	-	-	-	-	-	-	rw	rwh	rw	r	w	rw		

Bit	Function
bit(field)name	Explanation of bit(field)name *Description of the functions controlled by the different possible values of this bit(field).*

Elements:

REG_NAME	Short name of this register
A16 / A8	Long 16-bit address / Short 8-bit address
SFR/**ESFR**/**XReg**	Register space (SFR, ESFR or External/XBUS Register)
(* *) * *	Register contents after reset **0/1**: defined value, ’**X**’: undefined, ’**U**’: unchanged (undefined (’**X**’) after power up)
r/w	Access modes: can be read and/or write
▨ *h	Bits that are set/cleared by hardware are marked with a shaded access box and an ’**h**’ in it.

23.2 CPU General Purpose Registers (GPRs)

The GPRs form the register bank that the CPU works with. This register bank may be located anywhere within the internal RAM via the Context Pointer (CP). Due to the addressing mechanism, GPR banks can only reside within the internal RAM.
All GPRs are bit-addressable.

Table 23-1 General Purpose Word Registers

Name	Physical Address	8-Bit Address	Description	Reset Value
R0	(CP) + 0	$F0_H$	CPU General Purpose (Word) Reg. R0	$UUUU_H$
R1	(CP) + 2	$F1_H$	CPU General Purpose (Word) Reg. R1	$UUUU_H$
R2	(CP) + 4	$F2_H$	CPU General Purpose (Word) Reg. R2	$UUUU_H$
R3	(CP) + 6	$F3_H$	CPU General Purpose (Word) Reg. R3	$UUUU_H$
R4	(CP) + 8	$F4_H$	CPU General Purpose (Word) Reg. R4	$UUUU_H$
R5	(CP) + 10	$F5_H$	CPU General Purpose (Word) Reg. R5	$UUUU_H$
R6	(CP) + 12	$F6_H$	CPU General Purpose (Word) Reg. R6	$UUUU_H$
R7	(CP) + 14	$F7_H$	CPU General Purpose (Word) Reg. R7	$UUUU_H$
R8	(CP) + 16	$F8_H$	CPU General Purpose (Word) Reg. R8	$UUUU_H$
R9	(CP) + 18	$F9_H$	CPU General Purpose (Word) Reg. R9	$UUUU_H$
R10	(CP) + 20	FA_H	CPU General Purpose (Word) Reg. R10	$UUUU_H$
R11	(CP) + 22	FB_H	CPU General Purpose (Word) Reg. R11	$UUUU_H$
R12	(CP) + 24	FC_H	CPU General Purpose (Word) Reg. R12	$UUUU_H$
R13	(CP) + 26	FD_H	CPU General Purpose (Word) Reg. R13	$UUUU_H$
R14	(CP) + 28	FE_H	CPU General Purpose (Word) Reg. R14	$UUUU_H$
R15	(CP) + 30	FF_H	CPU General Purpose (Word) Reg. R15	$UUUU_H$

The first 8 GPRs (R7...R0) may also be accessed bytewise. Other than with SFRs, writing to a GPR byte does not affect the other byte of the respective GPR.
The respective halves of the byte-accessible registers receive special names:

Table 23-2 General Purpose Byte Registers

Name	Physical Address	8-Bit Address	Description	Reset Value
RL0	(CP) + 0	$F0_H$	CPU General Purpose (Byte) Reg. RL0	UU_H
RH0	(CP) + 1	$F1_H$	CPU General Purpose (Byte) Reg. RH0	UU_H
RL1	(CP) + 2	$F2_H$	CPU General Purpose (Byte) Reg. RL1	UU_H
RH1	(CP) + 3	$F3_H$	CPU General Purpose (Byte) Reg. RH1	UU_H
RL2	(CP) + 4	$F4_H$	CPU General Purpose (Byte) Reg. RL2	UU_H
RH2	(CP) + 5	$F5_H$	CPU General Purpose (Byte) Reg. RH2	UU_H
RL3	(CP) + 6	$F6_H$	CPU General Purpose (Byte) Reg. RL3	UU_H
RH3	(CP) + 7	$F7_H$	CPU General Purpose (Byte) Reg. RH3	UU_H
RL4	(CP) + 8	$F8_H$	CPU General Purpose (Byte) Reg. RL4	UU_H
RH4	(CP) + 9	$F9_H$	CPU General Purpose (Byte) Reg. RH4	UU_H
RL5	(CP) + 10	FA_H	CPU General Purpose (Byte) Reg. RL5	UU_H
RH5	(CP) + 11	FB_H	CPU General Purpose (Byte) Reg. RH5	UU_H
RL6	(CP) + 12	FC_H	CPU General Purpose (Byte) Reg. RL6	UU_H
RH6	(CP) + 13	FD_H	CPU General Purpose (Byte) Reg. RH6	UU_H
RL7	(CP) + 14	FE_H	CPU General Purpose (Byte) Reg. RL7	UU_H
RH7	(CP) + 15	FF_H	CPU General Purpose (Byte) Reg. RH7	UU_H

23.3 Special Function Registers ordered by Name

The following table lists all SFRs which are implemented in the C164 in alphabetical order.
Bit-addressable SFRs are marked with the letter “**b**” in column “Name”.
SFRs within the **Extended SFR-Space** (ESFRs) are marked with the letter “**E**” in column “Physical Address”. Registers within on-chip X-Peripherals are marked with the letter “**X**” in column “Physical Address”.

Table 23-3 C164 Registers, Ordered by Name

Name	Physical Address	8-Bit Addr.	Description	Reset Value
ADCIC **b**	FF98$_H$	CC$_H$	A/D Converter End of Conversion Interrupt Control Register	0000$_H$
ADCON **b**	FFA0$_H$	D0$_H$	A/D Converter Control Register	0000$_H$
ADDAT	FEA0$_H$	50$_H$	A/D Converter Result Register	0000$_H$
ADDAT2	F0A0$_H$ **E**	50$_H$	A/D Converter 2 Result Register	0000$_H$
ADDRSEL1	FE18$_H$	0C$_H$	Address Select Register 1	0000$_H$
ADDRSEL2	FE1A$_H$	0D$_H$	Address Select Register 2	0000$_H$
ADDRSEL3	FE1C$_H$	0E$_H$	Address Select Register 3	0000$_H$
ADDRSEL4	FE1E$_H$	0F$_H$	Address Select Register 4	0000$_H$
ADEIC **b**	FF9A$_H$	CD$_H$	A/D Converter Overrun Error Interrupt Control Register	0000$_H$
BUSCON0 **b**	FF0C$_H$	86$_H$	Bus Configuration Register 0	0000$_H$
BUSCON1 **b**	FF14$_H$	8A$_H$	Bus Configuration Register 1	0000$_H$
BUSCON2 **b**	FF16$_H$	8B$_H$	Bus Configuration Register 2	0000$_H$
BUSCON3 **b**	FF18$_H$	8C$_H$	Bus Configuration Register 3	0000$_H$
BUSCON4 **b**	FF1A$_H$	8D$_H$	Bus Configuration Register 4	0000$_H$
C1BTR	EF04$_H$ **X**	---	CAN1 Bit Timing Register	UUUU$_H$
C1CSR	EF00$_H$ **X**	---	CAN1 Control / Status Register	XX01$_H$
C1GMS	EF06$_H$ **X**	---	CAN1 Global Mask Short	UFUU$_H$
C1LARn	EFn4$_H$ **X**	---	CAN Lower Arbitration Register (msg. **n)**	UUUU$_H$
C1LGML	EF0A$_H$ **X**	---	CAN Lower Global Mask Long	UUUU$_H$
C1LMLM	EF0E$_H$ **X**	---	CAN Lower Mask of Last Message	UUUU$_H$
C1MCFGn	EFn6$_H$ **X**	---	CAN Message Configuration Register (msg. **n**)	UU$_H$

Table 23-3 C164 Registers, Ordered by Name (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
C1MCRn	EFn0$_H$ **X**	---	CAN Message Control Register (msg. **n**)	UUUU$_H$
C1PCIR	EF02$_H$ **X**	---	CAN1 Port Control / Interrupt Register	XXXX$_H$
C1UARn	EFn2$_H$ **X**	---	CAN Upper Arbitration Register (msg. **n**)	UUUU$_H$
C1UGML	EF08$_H$ **X**	---	CAN Upper Global Mask Long	UUUU$_H$
C1UMLM	EF0C$_H$ **X**	---	CAN Upper Mask of Last Message	UUUU$_H$
CC10IC b	FF8C$_H$	C6$_H$	External Interrupt 2 Control Register	0000$_H$
CC11IC b	FF8E$_H$	C7$_H$	External Interrupt 3 Control Register	0000$_H$
CC16	FE60$_H$	30$_H$	CAPCOM Register 16	0000$_H$
CC16IC b	F160$_H$ **E**	B0$_H$	CAPCOM Reg. 16 Interrupt Ctrl. Reg.	0000$_H$
CC17	FE62$_H$	31$_H$	CAPCOM Register 17	0000$_H$
CC17IC b	F162$_H$ **E**	B1$_H$	CAPCOM Reg. 17 Interrupt Ctrl. Reg.	0000$_H$
CC18	FE64$_H$	32$_H$	CAPCOM Register 18	0000$_H$
CC18IC b	F164$_H$ **E**	B2$_H$	CAPCOM Reg. 18 Interrupt Ctrl. Reg.	0000$_H$
CC19	FE66$_H$	33$_H$	CAPCOM Register 19	0000$_H$
CC19IC b	F166$_H$ **E**	B3$_H$	CAPCOM Reg. 19 Interrupt Ctrl. Reg.	0000$_H$
CC20	FE68$_H$	34$_H$	CAPCOM Register 20	0000$_H$
CC20IC b	F168$_H$ **E**	B4$_H$	CAPCOM Reg. 20 Interrupt Ctrl. Reg.	0000$_H$
CC21	FE6A$_H$	35$_H$	CAPCOM Register 21	0000$_H$
CC21IC b	F16A$_H$ **E**	B5$_H$	CAPCOM Reg. 21 Interrupt Ctrl. Reg.	0000$_H$
CC22	FE6C$_H$	36$_H$	CAPCOM Register 22	0000$_H$
CC22IC b	F16C$_H$ **E**	B6$_H$	CAPCOM Reg. 22 Interrupt Ctrl. Reg.	0000$_H$
CC23	FE6E$_H$	37$_H$	CAPCOM Register 23	0000$_H$
CC23IC b	F16E$_H$ **E**	B7$_H$	CAPCOM Reg. 23 Interrupt Ctrl. Reg.	0000$_H$
CC24	FE70$_H$	38$_H$	CAPCOM Register 24	0000$_H$
CC24IC b	F170$_H$ **E**	B8$_H$	CAPCOM Reg. 24 Interrupt Ctrl. Reg.	0000$_H$
CC25	FE72$_H$	39$_H$	CAPCOM Register 25	0000$_H$
CC25IC b	F172$_H$ **E**	B9$_H$	CAPCOM Reg. 25 Interrupt Ctrl. Reg.	0000$_H$
CC26	FE74$_H$	3A$_H$	CAPCOM Register 26	0000$_H$
CC26IC b	F174$_H$ **E**	BA$_H$	CAPCOM Reg. 26 Interrupt Ctrl. Reg.	0000$_H$
CC27	FE76$_H$	3B$_H$	CAPCOM Register 27	0000$_H$

Table 23-3 **C164 Registers, Ordered by Name** (cont'd)

Name		Physical Address		8-Bit Addr.	Description	Reset Value
CC27IC	**b**	F176$_H$	**E**	BB$_H$	CAPCOM Reg. 27 Interrupt Ctrl. Reg.	0000$_H$
CC28		FE78$_H$		3C$_H$	CAPCOM Register 28	0000$_H$
CC28IC	**b**	F178$_H$	**E**	BC$_H$	CAPCOM Reg. 28 Interrupt Ctrl. Reg.	0000$_H$
CC29		FE7A$_H$		3D$_H$	CAPCOM Register 29	0000$_H$
CC29IC	**b**	F184$_H$	**E**	C2$_H$	CAPCOM Reg. 29 Interrupt Ctrl. Reg.	0000$_H$
CC30		FE7C$_H$		3E$_H$	CAPCOM Register 30	0000$_H$
CC30IC	**b**	F18C$_H$	**E**	C6$_H$	CAPCOM Reg. 30 Interrupt Ctrl. Reg.	0000$_H$
CC31		FE7E$_H$		3F$_H$	CAPCOM Register 31	0000$_H$
CC31IC	**b**	F194$_H$	**E**	CA$_H$	CAPCOM Reg. 31 Interrupt Ctrl. Reg.	0000$_H$
CC60		FE30$_H$		18$_H$	CAPCOM 6 Register 0	0000$_H$
CC61		FE32$_H$		19$_H$	CAPCOM 6 Register 1	0000$_H$
CC62		FE34$_H$		1A$_H$	CAPCOM 6 Register 2	0000$_H$
CC6EIC	**b**	F188$_H$	**E**	C4$_H$	CAPCOM 6 Emergency Interrrupt Control Register	0000$_H$
CC6CIC	**b**	F17E$_H$	**E**	BF$_H$	CAPCOM 6 Interrupt Control Register	0000$_H$
CC6MCON	**b**	FF32$_H$		99$_H$	CAPCOM 6 Mode Control Register	00FF$_H$
CC6MIC	**b**	FF36$_H$		9B$_H$	CAPCOM 6 Mode Interrupt Ctrl. Reg.	0000$_H$
CC6MSEL		F036$_H$	**E**	1B$_H$	CAPCOM 6 Mode Select Register	0000$_H$
CC8IC	**b**	FF88$_H$		C4$_H$	External Interrupt 0 Control Register	0000$_H$
CC9IC	**b**	FF8A$_H$		C5$_H$	External Interrupt 1 Control Register	0000$_H$
CCM4	**b**	FF22$_H$		91$_H$	CAPCOM Mode Control Register 4	0000$_H$
CCM5	**b**	FF24$_H$		92$_H$	CAPCOM Mode Control Register 5	0000$_H$
CCM6	**b**	FF26$_H$		93$_H$	CAPCOM Mode Control Register 6	0000$_H$
CCM7	**b**	FF28$_H$		94$_H$	CAPCOM Mode Control Register 7	0000$_H$
CMP13		FE36$_H$		1B$_H$	CAPCOM 6 Timer 13 Compare Reg.	0000$_H$
CP		FE10$_H$		08$_H$	CPU Context Pointer Register	FC00$_H$
CSP		FE08$_H$		04$_H$	CPU Code Segment Pointer Register (8 bits, not directly writeable)	0000$_H$
CTCON	**b**	FF30$_H$		98$_H$	CAPCOM 6 Compare Timer Ctrl. Reg.	1010$_H$
DP0H	**b**	F102$_H$	**E**	81$_H$	P0H Direction Control Register	00$_H$

Table 23-3 C164 Registers, Ordered by Name (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
DP0L b	F100$_H$ **E**	80$_H$	P0L Direction Control Register	00$_H$
DP1H b	F106$_H$ **E**	83$_H$	P1H Direction Control Register	00$_H$
DP1L b	F104$_H$ **E**	82$_H$	P1L Direction Control Register	00$_H$
DP3 b	FFC6$_H$	E3$_H$	Port 3 Direction Control Register	0000$_H$
DP4 b	FFCA$_H$	E5$_H$	Port 4 Direction Control Register	00$_H$
DP8 b	FFD6$_H$	EB$_H$	Port 8 Direction Control Register	00$_H$
DPP0	FE00$_H$	00$_H$	CPU Data Page Pointer 0 Reg. (10 bits)	0000$_H$
DPP1	FE02$_H$	01$_H$	CPU Data Page Pointer 1 Reg. (10 bits)	0001$_H$
DPP2	FE04$_H$	02$_H$	CPU Data Page Pointer 2 Reg. (10 bits)	0002$_H$
DPP3	FE06$_H$	03$_H$	CPU Data Page Pointer 3 Reg. (10 bits)	0003$_H$
EXICON b	F1C0$_H$ **E**	E0$_H$	External Interrupt Control Register	0000$_H$
EXISEL b	F1DA$_H$ **E**	ED$_H$	External Interrupt Source Select Reg.	0000$_H$
FOCON b	FFAA$_H$	D5$_H$	Frequency Output Control Register	0000$_H$
IDCHIP	F07C$_H$ **E**	3E$_H$	Identifier	XXXX$_H$
IDMANUF	F07E$_H$ **E**	3F$_H$	Identifier	1820$_H$
IDMEM	F07A$_H$ **E**	3D$_H$	Identifier	XXXX$_H$
IDPROG	F078$_H$ **E**	3C$_H$	Identifier	XXXX$_H$
IDMEM2	F076$_H$ **E**	3B$_H$	Identifier	XXXX$_H$
ISNC b	F1DE$_H$ **E**	EF$_H$	Interrupt Subnode Control Register	0000$_H$
MDC b	FF0E$_H$	87$_H$	CPU Multiply Divide Control Register	0000$_H$
MDH	FE0C$_H$	06$_H$	CPU Multiply Divide Reg. – High Word	0000$_H$
MDL	FE0E$_H$	07$_H$	CPU Multiply Divide Reg. – Low Word	0000$_H$
ODP3 b	F1C6$_H$ **E**	E3$_H$	Port 3 Open Drain Control Register	0000$_H$
ODP4 b	F1CA$_H$ **E**	E5$_H$	Port 4 Open Drain Control Register	00$_H$
ODP8 b	F1D6$_H$ **E**	EB$_H$	Port 8 Open Drain Control Register	00$_H$
ONES b	FF1E$_H$	8F$_H$	Constant Value 1's Register (read only)	FFFF$_H$
P0H b	FF02$_H$	81$_H$	Port 0 High Reg. (Upper half of PORT0)	00$_H$
P0L b	FF00$_H$	80$_H$	Port 0 Low Reg. (Lower half of PORT0)	00$_H$
P1H b	FF06$_H$	83$_H$	Port 1 High Reg. (Upper half of PORT1)	00$_H$
P1L b	FF04$_H$	82$_H$	Port 1 Low Reg. (Lower half of PORT1)	00$_H$

Table 23-3 C164 Registers, Ordered by Name (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
P3 **b**	FFC4$_H$	E2$_H$	Port 3 Register	0000$_H$
P4 **b**	FFC8$_H$	E4$_H$	Port 4 Register (7 bits)	00$_H$
P5 **b**	FFA2$_H$	D1$_H$	Port 5 Register (read only)	XXXX$_H$
P5DIDIS **b**	FFA4$_H$	D2$_H$	Port 5 Digital Input Disable Register	0000$_H$
P8 **b**	FFD4$_H$	EA$_H$	Port 8 Register (8 bits)	00$_H$
PECC0	FEC0$_H$	60$_H$	PEC Channel 0 Control Register	0000$_H$
PECC1	FEC2$_H$	61$_H$	PEC Channel 1 Control Register	0000$_H$
PECC2	FEC4$_H$	62$_H$	PEC Channel 2 Control Register	0000$_H$
PECC3	FEC6$_H$	63$_H$	PEC Channel 3 Control Register	0000$_H$
PECC4	FEC8$_H$	64$_H$	PEC Channel 4 Control Register	0000$_H$
PECC5	FECA$_H$	65$_H$	PEC Channel 5 Control Register	0000$_H$
PECC6	FECC$_H$	66$_H$	PEC Channel 6 Control Register	0000$_H$
PECC7	FECE$_H$	67$_H$	PEC Channel 7 Control Register	0000$_H$
PICON **b**	F1C4$_H$ **E**	E2$_H$	Port Input Threshold Control Register	0000$_H$
POCON0H	F082$_H$ **E**	41$_H$	Port P0H Output Control Register	0000$_H$
POCON0L	F080$_H$ **E**	40$_H$	Port P0L Output Control Register	0000$_H$
POCON1H	F086$_H$ **E**	43$_H$	Port P1H Output Control Register	0000$_H$
POCON1L	F084$_H$ **E**	42$_H$	Port P1L Output Control Register	0000$_H$
POCON20	F0AA$_H$ **E**	55$_H$	Dedicated Pin Output Control Register	0000$_H$
POCON3	F08A$_H$ **E**	45$_H$	Port P3 Output Control Register	0000$_H$
POCON4	F08C$_H$ **E**	46$_H$	Port P4 Output Control Register	0000$_H$
POCON8	F092$_H$ **E**	49$_H$	Port P8 Output Control Register	0000$_H$
PSW **b**	FF10$_H$	88$_H$	CPU Program Status Word	0000$_H$
PTCR	F0AE$_H$ **E**	57$_H$	Port Temperature Compensation Reg.	0000$_H$
RP0H **b**	F108$_H$ **E**	84$_H$	System Startup Config. Reg. (Rd. only)	XX$_H$
RSTCON **b**	F1E0$_H$ **m**	---	Reset Control Register	00XX$_H$
RTCH	F0D6$_H$ **E**	6B$_H$	RTC High Register	no
RTCL	F0D4$_H$ **E**	6A$_H$	RTC Low Register	no
S0BG	FEB4$_H$	5A$_H$	Serial Channel 0 Baud Rate Generator Reload Register	0000$_H$

Table 23-3 C164 Registers, Ordered by Name (cont'd)

Name		Physical Address		8-Bit Addr.	Description	Reset Value
S0CON	**b**	FFB0$_H$		D8$_H$	Serial Channel 0 Control Register	0000$_H$
S0EIC	**b**	FF70$_H$		B8$_H$	Serial Channel 0 Error Interrupt Ctrl. Reg.	0000$_H$
S0RBUF		FEB2$_H$		59$_H$	Serial Channel 0 Receive Buffer Reg. (read only)	XXXX$_H$
S0RIC	**b**	FF6E$_H$		B7$_H$	Serial Channel 0 Receive Interrupt Control Register	0000$_H$
S0TBIC	**b**	F19C$_H$	**E**	CE$_H$	Serial Channel 0 Transmit Buffer Interrupt Control Register	0000$_H$
S0TBUF		FEB0$_H$		58$_H$	Serial Channel 0 Transmit Buffer Reg. (write only)	0000$_H$
S0TIC	**b**	FF6C$_H$		B6$_H$	Serial Channel 0 Transmit Interrupt Control Register	0000$_H$
SP		FE12$_H$		09$_H$	CPU System Stack Pointer Register	FC00$_H$
SSCBR		F0B4$_H$	**E**	5A$_H$	SSC Baudrate Register	0000$_H$
SSCCON	**b**	FFB2$_H$		D9$_H$	SSC Control Register	0000$_H$
SSCEIC	**b**	FF76$_H$		BB$_H$	SSC Error Interrupt Control Register	0000$_H$
SSCRB		F0B2$_H$	**E**	59$_H$	SSC Receive Buffer	XXXX$_H$
SSCRIC	**b**	FF74$_H$		BA$_H$	SSC Receive Interrupt Control Register	0000$_H$
SSCTB		F0B0$_H$	**E**	58$_H$	SSC Transmit Buffer	0000$_H$
SSCTIC	**b**	FF72$_H$		B9$_H$	SSC Transmit Interrupt Control Register	0000$_H$
STKOV		FE14$_H$		0A$_H$	CPU Stack Overflow Pointer Register	FA00$_H$
STKUN		FE16$_H$		0B$_H$	CPU Stack Underflow Pointer Register	FC00$_H$
SYSCON	**b**	FF12$_H$		89$_H$	CPU System Configuration Register	[1] 0xx0$_H$
SYSCON1	**b**	F1DC$_H$	**E**	EE$_H$	CPU System Configuration Register 1	0000$_H$
SYSCON2	**b**	F1D0$_H$	**E**	E8$_H$	CPU System Configuration Register 2	0000$_H$
SYSCON3	**b**	F1D4$_H$	**E**	EA$_H$	CPU System Configuration Register 3	0000$_H$
T12IC	**b**	F190$_H$	**E**	C8$_H$	CAPCOM 6 Timer 12 Interrupt Ctrl. Reg.	0000$_H$
T12OF		F034$_H$	**E**	1A$_H$	CAPCOM 6 Timer 12 Offset Register	0000$_H$
T12P		F030$_H$	**E**	18$_H$	CAPCOM 6 Timer 12 Period Register	0000$_H$
T13IC	**b**	F198$_H$	**E**	CC$_H$	CAPCOM 6 Timer 13 Interrupt Ctrl. Reg.	0000$_H$

Table 23-3 C164 Registers, Ordered by Name (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
T13P	F032$_H$ **E**	19$_H$	CAPCOM 6 Timer 13 Period Register	0000$_H$
T14	F0D2$_H$ **E**	69$_H$	RTC Timer 14 Register	no
T14REL	F0D0$_H$ **E**	68$_H$	RTC Timer 14 Reload Register	no
T2	FE40$_H$	20$_H$	GPT1 Timer 2 Register	0000$_H$
T2CON b	FF40$_H$	A0$_H$	GPT1 Timer 2 Control Register	0000$_H$
T2IC b	FF60$_H$	B0$_H$	GPT1 Timer 2 Interrupt Control Register	0000$_H$
T3	FE42$_H$	21$_H$	GPT1 Timer 3 Register	0000$_H$
T3CON b	FF42$_H$	A1$_H$	GPT1 Timer 3 Control Register	0000$_H$
T3IC b	FF62$_H$	B1$_H$	GPT1 Timer 3 Interrupt Control Register	0000$_H$
T4	FE44$_H$	22$_H$	GPT1 Timer 4 Register	0000$_H$
T4CON b	FF44$_H$	A2$_H$	GPT1 Timer 4 Control Register	0000$_H$
T4IC b	FF64$_H$	B2$_H$	GPT1 Timer 4 Interrupt Control Register	0000$_H$
T7	F050$_H$ **E**	28$_H$	CAPCOM Timer 7 Register	0000$_H$
T78CON b	FF20$_H$	90$_H$	CAPCOM Timer 7 and 8 Ctrl. Reg.	0000$_H$
T7IC b	F17A$_H$ **E**	BD$_H$	CAPCOM Timer 7 Interrupt Ctrl. Reg.	0000$_H$
T7REL	F054$_H$ **E**	2A$_H$	CAPCOM Timer 7 Reload Register	0000$_H$
T8	F052$_H$ **E**	29$_H$	CAPCOM Timer 8 Register	0000$_H$
T8IC b	F17C$_H$ **E**	BE$_H$	CAPCOM Timer 8 Interrupt Ctrl. Reg.	0000$_H$
T8REL	F056$_H$ **E**	2B$_H$	CAPCOM Timer 8 Reload Register	0000$_H$
TFR b	FFAC$_H$	D6$_H$	Trap Flag Register	0000$_H$
TRCON b	FF34$_H$	9A$_H$	CAPCOM 6 Trap Enable Ctrl. Reg.	00XX$_H$
WDT	FEAE$_H$	57$_H$	Watchdog Timer Register (read only)	0000$_H$
WDTCON	FFAE$_H$	D7$_H$	Watchdog Timer Control Register	[2] 00xx$_H$
XP0IC b	F186$_H$ **E**	C3$_H$	CAN1 Module Interrupt Control Register	0000$_H$
XP1IC b	F18E$_H$ **E**	C7$_H$	Flash Termination Interrupt Control Reg.	0000$_H$
XP3IC b	F19E$_H$ **E**	CF$_H$	PLL/RTC Interrupt Control Register	0000$_H$
ZEROS b	FF1C$_H$	8E$_H$	Constant Value 0's Register (read only)	0000$_H$

[1] The system configuration is selected during reset.

[2] The reset value depends on the indicated reset source.

23.4 Registers ordered by Address

The following table lists all SFRs which are implemented in the C164 ordered by their physical address.
Bit-addressable SFRs are marked with the letter “**b**” in column “Name”.
SFRs within the **Extended SFR-Space** (ESFRs) are marked with the letter “**E**” in column “Physical Address”. Registers within on-chip X-Peripherals are marked with the letter “**X**” in column “Physical Address”.

Table 23-4 C164 Registers, Ordered by Name

Name	Physical Address		8-Bit Addr.	Description	Reset Value
C1CSR	EF00$_H$	**X**	---	CAN1 Control / Status Register	XX01$_H$
C1PCIR	EF02$_H$	**X**	---	CAN1 Port Control / Interrupt Register	XXXX$_H$
C1BTR	EF04$_H$	**X**	---	CAN1 Bit Timing Register	UUUU$_H$
C1GMS	EF06$_H$	**X**	---	CAN1 Global Mask Short	UFUU$_H$
C1UGML	EF08$_H$	**X**	---	CAN Upper Global Mask Long	UUUU$_H$
C1LGML	EF0A$_H$	**X**	---	CAN Lower Global Mask Long	UUUU$_H$
C1UMLM	EF0C$_H$	**X**	---	CAN Upper Mask of Last Message	UUUU$_H$
C1LMLM	EF0E$_H$	**X**	---	CAN Lower Mask of Last Message	UUUU$_H$
C1MCRn	EFn0$_H$	**X**	---	CAN Message Control Register (msg. **n**)	UUUU$_H$
C1UARn	EFn2$_H$	**X**	---	CAN Upper Arbitration Register (msg. **n**)	UUUU$_H$
C1LARn	EFn4$_H$	**X**	---	CAN Lower Arbitration Register (msg. **n)**	UUUU$_H$
C1MCFGn	EFn6$_H$	**X**	---	CAN Message Configuration Register (msg. **n**)	UU$_H$
T12P	F030$_H$	**E**	18$_H$	CAPCOM 6 Timer 12 Period Register	0000$_H$
T13P	F032$_H$	**E**	19$_H$	CAPCOM 6 Timer 13 Period Register	0000$_H$
T12OF	F034$_H$	**E**	1A$_H$	CAPCOM 6 Timer 12 Offset Register	0000$_H$
CC6MSEL	F036$_H$	**E**	1B$_H$	CAPCOM 6 Mode Select Register	0000$_H$
T7	F050$_H$	**E**	28$_H$	CAPCOM Timer 7 Register	0000$_H$
T8	F052$_H$	**E**	29$_H$	CAPCOM Timer 8 Register	0000$_H$
T7REL	F054$_H$	**E**	2A$_H$	CAPCOM Timer 7 Reload Register	0000$_H$
T8REL	F056$_H$	**E**	2B$_H$	CAPCOM Timer 8 Reload Register	0000$_H$
IDMEM2	F076$_H$	**E**	3B$_H$	Identifier	XXXX$_H$
IDPROG	F078$_H$	**E**	3C$_H$	Identifier	XXXX$_H$

Table 23-4 **C164 Registers, Ordered by Name** (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
IDMEM	F07A$_H$ **E**	3D$_H$	Identifier	XXXX$_H$
IDCHIP	F07C$_H$ **E**	3E$_H$	Identifier	XXXX$_H$
IDMANUF	F07E$_H$ **E**	3F$_H$	Identifier	1820$_H$
POCON0L	F080$_H$ **E**	40$_H$	Port P0L Output Control Register	0000$_H$
POCON0H	F082$_H$ **E**	41$_H$	Port P0H Output Control Register	0000$_H$
POCON1L	F084$_H$ **E**	42$_H$	Port P1L Output Control Register	0000$_H$
POCON1H	F086$_H$ **E**	43$_H$	Port P1H Output Control Register	0000$_H$
POCON3	F08A$_H$ **E**	45$_H$	Port P3 Output Control Register	0000$_H$
POCON4	F08C$_H$ **E**	46$_H$	Port P4 Output Control Register	0000$_H$
POCON8	F092$_H$ **E**	49$_H$	Port P8 Output Control Register	0000$_H$
ADDAT2	F0A0$_H$ **E**	50$_H$	A/D Converter 2 Result Register	0000$_H$
POCON20	F0AA$_H$ **E**	55$_H$	Dedicated Pin Output Control Register	0000$_H$
PTCR	F0AE$_H$ **E**	57$_H$	Port Temperature Compensation Reg.	0000$_H$
SSCTB	F0B0$_H$ **E**	58$_H$	SSC Transmit Buffer	0000$_H$
SSCRB	F0B2$_H$ **E**	59$_H$	SSC Receive Buffer	XXXX$_H$
SSCBR	F0B4$_H$ **E**	5A$_H$	SSC Baudrate Register	0000$_H$
T14REL	F0D0$_H$ **E**	68$_H$	RTC Timer 14 Reload Register	no
T14	F0D2$_H$ **E**	69$_H$	RTC Timer 14 Register	no
RTCL	F0D4$_H$ **E**	6A$_H$	RTC Low Register	no
RTCH	F0D6$_H$ **E**	6B$_H$	RTC High Register	no
DP0L b	F100$_H$ **E**	80$_H$	P0L Direction Control Register	00$_H$
DP0H b	F102$_H$ **E**	81$_H$	P0H Direction Control Register	00$_H$
DP1L b	F104$_H$ **E**	82$_H$	P1L Direction Control Register	00$_H$
DP1H b	F106$_H$ **E**	83$_H$	P1H Direction Control Register	00$_H$
RP0H b	F108$_H$ **E**	84$_H$	System Startup Config. Reg. (Rd. only)	XX$_H$
CC16IC b	F160$_H$ **E**	B0$_H$	CAPCOM Reg. 16 Interrupt Ctrl. Reg.	0000$_H$
CC17IC b	F162$_H$ **E**	B1$_H$	CAPCOM Reg. 17 Interrupt Ctrl. Reg.	0000$_H$
CC18IC b	F164$_H$ **E**	B2$_H$	CAPCOM Reg. 18 Interrupt Ctrl. Reg.	0000$_H$
CC19IC b	F166$_H$ **E**	B3$_H$	CAPCOM Reg. 19 Interrupt Ctrl. Reg.	0000$_H$
CC20IC b	F168$_H$ **E**	B4$_H$	CAPCOM Reg. 20 Interrupt Ctrl. Reg.	0000$_H$

Table 23-4 C164 Registers, Ordered by Name (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
CC21IC b	$F16A_H$ **E**	$B5_H$	CAPCOM Reg. 21 Interrupt Ctrl. Reg.	0000_H
CC22IC b	$F16C_H$ **E**	$B6_H$	CAPCOM Reg. 22 Interrupt Ctrl. Reg.	0000_H
CC23IC b	$F16E_H$ **E**	$B7_H$	CAPCOM Reg. 23 Interrupt Ctrl. Reg.	0000_H
CC24IC b	$F170_H$ **E**	$B8_H$	CAPCOM Reg. 24 Interrupt Ctrl. Reg.	0000_H
CC25IC b	$F172_H$ **E**	$B9_H$	CAPCOM Reg. 25 Interrupt Ctrl. Reg.	0000_H
CC26IC b	$F174_H$ **E**	BA_H	CAPCOM Reg. 26 Interrupt Ctrl. Reg.	0000_H
CC27IC b	$F176_H$ **E**	BB_H	CAPCOM Reg. 27 Interrupt Ctrl. Reg.	0000_H
CC28IC b	$F178_H$ **E**	BC_H	CAPCOM Reg. 28 Interrupt Ctrl. Reg.	0000_H
T7IC b	$F17A_H$ **E**	BD_H	CAPCOM Timer 7 Interrupt Ctrl. Reg.	0000_H
T8IC b	$F17C_H$ **E**	BE_H	CAPCOM Timer 8 Interrupt Ctrl. Reg.	0000_H
CC6CIC b	$F17E_H$ **E**	BF_H	CAPCOM 6 Interrupt Control Register	0000_H
CC29IC b	$F184_H$ **E**	$C2_H$	CAPCOM Reg. 29 Interrupt Ctrl. Reg.	0000_H
XP0IC b	$F186_H$ **E**	$C3_H$	CAN1 Module Interrupt Control Register	0000_H
CC6EIC b	$F188_H$ **E**	$C4_H$	CAPCOM 6 Emergency Interrrupt Control Register	0000_H
CC30IC b	$F18C_H$ **E**	$C6_H$	CAPCOM Reg. 30 Interrupt Ctrl. Reg.	0000_H
XP1IC b	$F18E_H$ **E**	$C7_H$	Flash Termination Interrupt Control Reg.	0000_H
T12IC b	$F190_H$ **E**	$C8_H$	CAPCOM 6 Timer 12 Interrupt Ctrl. Reg.	0000_H
CC31IC b	$F194_H$ **E**	CA_H	CAPCOM Reg. 31 Interrupt Ctrl. Reg.	0000_H
T13IC b	$F198_H$ **E**	CC_H	CAPCOM 6 Timer 13 Interrupt Ctrl. Reg.	0000_H
S0TBIC b	$F19C_H$ **E**	CE_H	Serial Channel 0 Transmit Buffer Interrupt Control Register	0000_H
XP3IC b	$F19E_H$ **E**	CF_H	PLL/RTC Interrupt Control Register	0000_H
EXICON b	$F1C0_H$ **E**	$E0_H$	External Interrupt Control Register	0000_H
PICON b	$F1C4_H$ **E**	$E2_H$	Port Input Threshold Control Register	0000_H
ODP3 b	$F1C6_H$ **E**	$E3_H$	Port 3 Open Drain Control Register	0000_H
ODP4 b	$F1CA_H$ **E**	$E5_H$	Port 4 Open Drain Control Register	00_H
SYSCON2 b	$F1D0_H$ **E**	$E8_H$	CPU System Configuration Register 2	0000_H
SYSCON3 b	$F1D4_H$ **E**	EA_H	CPU System Configuration Register 3	0000_H
ODP8 b	$F1D6_H$ **E**	EB_H	Port 8 Open Drain Control Register	00_H

Table 23-4 C164 Registers, Ordered by Name (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
EXISEL b	F1DA$_H$ **E**	ED$_H$	External Interrupt Source Select Reg.	0000$_H$
SYSCON1 b	F1DC$_H$ **E**	EE$_H$	CPU System Configuration Register 1	0000$_H$
ISNC b	F1DE$_H$ **E**	EF$_H$	Interrupt Subnode Control Register	0000$_H$
RSTCON b	F1E0$_H$ **m**	---	Reset Control Register	00XX$_H$
DPP0	FE00$_H$	00$_H$	CPU Data Page Pointer 0 Reg. (10 bits)	0000$_H$
DPP1	FE02$_H$	01$_H$	CPU Data Page Pointer 1 Reg. (10 bits)	0001$_H$
DPP2	FE04$_H$	02$_H$	CPU Data Page Pointer 2 Reg. (10 bits)	0002$_H$
DPP3	FE06$_H$	03$_H$	CPU Data Page Pointer 3 Reg. (10 bits)	0003$_H$
CSP	FE08$_H$	04$_H$	CPU Code Segment Pointer Register (8 bits, not directly writeable)	0000$_H$
MDH	FE0C$_H$	06$_H$	CPU Multiply Divide Reg. – High Word	0000$_H$
MDL	FE0E$_H$	07$_H$	CPU Multiply Divide Reg. – Low Word	0000$_H$
CP	FE10$_H$	08$_H$	CPU Context Pointer Register	FC00$_H$
SP	FE12$_H$	09$_H$	CPU System Stack Pointer Register	FC00$_H$
STKOV	FE14$_H$	0A$_H$	CPU Stack Overflow Pointer Register	FA00$_H$
STKUN	FE16$_H$	0B$_H$	CPU Stack Underflow Pointer Register	FC00$_H$
ADDRSEL1	FE18$_H$	0C$_H$	Address Select Register 1	0000$_H$
ADDRSEL2	FE1A$_H$	0D$_H$	Address Select Register 2	0000$_H$
ADDRSEL3	FE1C$_H$	0E$_H$	Address Select Register 3	0000$_H$
ADDRSEL4	FE1E$_H$	0F$_H$	Address Select Register 4	0000$_H$
CC60	FE30$_H$	18$_H$	CAPCOM 6 Register 0	0000$_H$
CC61	FE32$_H$	19$_H$	CAPCOM 6 Register 1	0000$_H$
CC62	FE34$_H$	1A$_H$	CAPCOM 6 Register 2	0000$_H$
CMP13	FE36$_H$	1B$_H$	CAPCOM 6 Timer 13 Compare Reg.	0000$_H$
T2	FE40$_H$	20$_H$	GPT1 Timer 2 Register	0000$_H$
T3	FE42$_H$	21$_H$	GPT1 Timer 3 Register	0000$_H$
T4	FE44$_H$	22$_H$	GPT1 Timer 4 Register	0000$_H$
CC16	FE60$_H$	30$_H$	CAPCOM Register 16	0000$_H$
CC17	FE62$_H$	31$_H$	CAPCOM Register 17	0000$_H$
CC18	FE64$_H$	32$_H$	CAPCOM Register 18	0000$_H$

Table 23-4 C164 Registers, Ordered by Name (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
CC19	$FE66_H$	33_H	CAPCOM Register 19	0000_H
CC20	$FE68_H$	34_H	CAPCOM Register 20	0000_H
CC21	$FE6A_H$	35_H	CAPCOM Register 21	0000_H
CC22	$FE6C_H$	36_H	CAPCOM Register 22	0000_H
CC23	$FE6E_H$	37_H	CAPCOM Register 23	0000_H
CC24	$FE70_H$	38_H	CAPCOM Register 24	0000_H
CC25	$FE72_H$	39_H	CAPCOM Register 25	0000_H
CC26	$FE74_H$	$3A_H$	CAPCOM Register 26	0000_H
CC27	$FE76_H$	$3B_H$	CAPCOM Register 27	0000_H
CC28	$FE78_H$	$3C_H$	CAPCOM Register 28	0000_H
CC29	$FE7A_H$	$3D_H$	CAPCOM Register 29	0000_H
CC30	$FE7C_H$	$3E_H$	CAPCOM Register 30	0000_H
CC31	$FE7E_H$	$3F_H$	CAPCOM Register 31	0000_H
ADDAT	$FEA0_H$	50_H	A/D Converter Result Register	0000_H
WDT	$FEAE_H$	57_H	Watchdog Timer Register (read only)	0000_H
S0TBUF	$FEB0_H$	58_H	Serial Channel 0 Transmit Buffer Reg. (write only)	0000_H
S0RBUF	$FEB2_H$	59_H	Serial Channel 0 Receive Buffer Reg. (read only)	$XXXX_H$
S0BG	$FEB4_H$	$5A_H$	Serial Channel 0 Baud Rate Generator Reload Register	0000_H
PECC0	$FEC0_H$	60_H	PEC Channel 0 Control Register	0000_H
PECC1	$FEC2_H$	61_H	PEC Channel 1 Control Register	0000_H
PECC2	$FEC4_H$	62_H	PEC Channel 2 Control Register	0000_H
PECC3	$FEC6_H$	63_H	PEC Channel 3 Control Register	0000_H
PECC4	$FEC8_H$	64_H	PEC Channel 4 Control Register	0000_H
PECC5	$FECA_H$	65_H	PEC Channel 5 Control Register	0000_H
PECC6	$FECC_H$	66_H	PEC Channel 6 Control Register	0000_H
PECC7	$FECE_H$	67_H	PEC Channel 7 Control Register	0000_H
P0L b	$FF00_H$	80_H	Port 0 Low Reg. (Lower half of PORT0)	00_H

Table 23-4 C164 Registers, Ordered by Name (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
P0H b	$FF02_H$	81_H	Port 0 High Reg. (Upper half of PORT0)	00_H
P1L b	$FF04_H$	82_H	Port 1 Low Reg. (Lower half of PORT1)	00_H
P1H b	$FF06_H$	83_H	Port 1 High Reg. (Upper half of PORT1)	00_H
BUSCON0 b	$FF0C_H$	86_H	Bus Configuration Register 0	0000_H
MDC b	$FF0E_H$	87_H	CPU Multiply Divide Control Register	0000_H
PSW b	$FF10_H$	88_H	CPU Program Status Word	0000_H
SYSCON b	$FF12_H$	89_H	CPU System Configuration Register	[1] $0xx0_H$
BUSCON1 b	$FF14_H$	$8A_H$	Bus Configuration Register 1	0000_H
BUSCON2 b	$FF16_H$	$8B_H$	Bus Configuration Register 2	0000_H
BUSCON3 b	$FF18_H$	$8C_H$	Bus Configuration Register 3	0000_H
BUSCON4 b	$FF1A_H$	$8D_H$	Bus Configuration Register 4	0000_H
ZEROS b	$FF1C_H$	$8E_H$	Constant Value 0's Register (read only)	0000_H
ONES b	$FF1E_H$	$8F_H$	Constant Value 1's Register (read only)	$FFFF_H$
T78CON b	$FF20_H$	90_H	CAPCOM Timer 7 and 8 Ctrl. Reg.	0000_H
CCM4 b	$FF22_H$	91_H	CAPCOM Mode Control Register 4	0000_H
CCM5 b	$FF24_H$	92_H	CAPCOM Mode Control Register 5	0000_H
CCM6 b	$FF26_H$	93_H	CAPCOM Mode Control Register 6	0000_H
CCM7 b	$FF28_H$	94_H	CAPCOM Mode Control Register 7	0000_H
CTCON b	$FF30_H$	98_H	CAPCOM 6 Compare Timer Ctrl. Reg.	1010_H
CC6MCON b	$FF32_H$	99_H	CAPCOM 6 Mode Control Register	$00FF_H$
TRCON b	$FF34_H$	$9A_H$	CAPCOM 6 Trap Enable Ctrl. Reg.	$00XX_H$
CC6MIC b	$FF36_H$	$9B_H$	CAPCOM 6 Mode Interrupt Ctrl. Reg.	0000_H
T2CON b	$FF40_H$	$A0_H$	GPT1 Timer 2 Control Register	0000_H
T3CON b	$FF42_H$	$A1_H$	GPT1 Timer 3 Control Register	0000_H
T4CON b	$FF44_H$	$A2_H$	GPT1 Timer 4 Control Register	0000_H
T2IC b	$FF60_H$	$B0_H$	GPT1 Timer 2 Interrupt Control Register	0000_H
T3IC b	$FF62_H$	$B1_H$	GPT1 Timer 3 Interrupt Control Register	0000_H
T4IC b	$FF64_H$	$B2_H$	GPT1 Timer 4 Interrupt Control Register	0000_H
S0TIC b	$FF6C_H$	$B6_H$	Serial Channel 0 Transmit Interrupt Control Register	0000_H

Table 23-4 C164 Registers, Ordered by Name (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
S0RIC **b**	FF6E$_H$	B7$_H$	Serial Channel 0 Receive Interrupt Control Register	0000$_H$
S0EIC **b**	FF70$_H$	B8$_H$	Serial Channel 0 Error Interrupt Ctrl. Reg.	0000$_H$
SSCTIC **b**	FF72$_H$	B9$_H$	SSC Transmit Interrupt Control Register	0000$_H$
SSCRIC **b**	FF74$_H$	BA$_H$	SSC Receive Interrupt Control Register	0000$_H$
SSCEIC **b**	FF76$_H$	BB$_H$	SSC Error Interrupt Control Register	0000$_H$
CC8IC **b**	FF88$_H$	C4$_H$	External Interrupt 0 Control Register	0000$_H$
CC9IC **b**	FF8A$_H$	C5$_H$	External Interrupt 1 Control Register	0000$_H$
CC10IC **b**	FF8C$_H$	C6$_H$	External Interrupt 2 Control Register	0000$_H$
CC11IC **b**	FF8E$_H$	C7$_H$	External Interrupt 3 Control Register	0000$_H$
ADCIC **b**	FF98$_H$	CC$_H$	A/D Converter End of Conversion Interrupt Control Register	0000$_H$
ADEIC **b**	FF9A$_H$	CD$_H$	A/D Converter Overrun Error Interrupt Control Register	0000$_H$
ADCON **b**	FFA0$_H$	D0$_H$	A/D Converter Control Register	0000$_H$
P5 **b**	FFA2$_H$	D1$_H$	Port 5 Register (read only)	XXXX$_H$
P5DIDIS **b**	FFA4$_H$	D2$_H$	Port 5 Digital Input Disable Register	0000$_H$
FOCON **b**	FFAA$_H$	D5$_H$	Frequency Output Control Register	0000$_H$
TFR **b**	FFAC$_H$	D6$_H$	Trap Flag Register	0000$_H$
WDTCON	FFAE$_H$	D7$_H$	Watchdog Timer Control Register	[2] 00xx$_H$
S0CON **b**	FFB0$_H$	D8$_H$	Serial Channel 0 Control Register	0000$_H$
SSCCON **b**	FFB2$_H$	D9$_H$	SSC Control Register	0000$_H$
P3 **b**	FFC4$_H$	E2$_H$	Port 3 Register	0000$_H$
DP3 **b**	FFC6$_H$	E3$_H$	Port 3 Direction Control Register	0000$_H$
P4 **b**	FFC8$_H$	E4$_H$	Port 4 Register (7 bits)	00$_H$
DP4 **b**	FFCA$_H$	E5$_H$	Port 4 Direction Control Register	00$_H$
P8 **b**	FFD4$_H$	EA$_H$	Port 8 Register (8 bits)	00$_H$
DP8 **b**	FFD6$_H$	EB$_H$	Port 8 Direction Control Register	00$_H$

[1] The system configuration is selected during reset.

[2] The reset value depends on the indicated reset source.

23.5 Special Notes

PEC Pointer Registers

The source and destination pointers for the peripheral event controller are mapped to a special area within the internal RAM. Pointers that are not occupied by the PEC may therefore be used like normal RAM. During Power Down mode or any warm reset the PEC pointers are preserved.

The PEC and its registers are described in chapter “Interrupt and Trap Functions”.

GPR Access in the ESFR Area

The locations 00’F000$_H$...00’F01E$_H$ within the ESFR area are reserved and allow to access the current register bank via short register addressing modes. The GPRs are mirrored to the ESFR area which allows access to the current register bank even after switching register spaces (see example below).

```
MOV       R5, DP3           ;GPR access via SFR area
EXTR      #1
MOV       R5, ODP3          ;GPR access via ESFR area
```

Writing Bytes to SFRs

All special function registers may be accessed wordwise or bytewise (some of them even bitwise). Reading bytes from word SFRs is a non-critical operation. However, when writing bytes to word SFRs the complementary byte of the respective SFR is cleared with the write operation.

24 Instruction Set Summary

This chapter briefly summarizes the C164's instructions ordered by instruction classes. This provides a basic understanding of the C164's instruction set, the power and versatility of the instructions and their general usage.

A detailed description of each single instruction, including its operand data type, condition flag settings, addressing modes, length (number of bytes) and object code format is provided in the **"Instruction Set Manual"** for the C166 Family. This manual also provides tables ordering the instructions according to various criteria, to allow quick references.

Summary of Instruction Classes

Grouping the various instruction into classes aids in identifying similar instructions (e.g. SHR, ROR) and variations of certain instructions (e.g. ADD, ADDB). This provides an easy access to the possibilities and the power of the instructions of the C164.

Note: The used mnemonics refer to the detailled description.

Arithmetic Instructions

• Addition of two words or bytes:	ADD	ADDB
• Addition with Carry of two words or bytes:	ADDC	ADDCB
• Subtraction of two words or bytes:	SUB	SUBB
• Subtraction with Carry of two words or bytes:	SUBC	SUBCB
• 16*16 bit signed or unsigned multiplication:	MUL	MULU
• 16/16 bit signed or unsigned division:	DIV	DIVU
• 32/16 bit signed or unsigned division:	DIVL	DIVLU
• 1's complement of a word or byte:	CPL	CPLB
• 2's complement (negation) of a word or byte:	NEG	NEGB

Logical Instructions

• Bitwise ANDing of two words or bytes:	AND	ANDB
• Bitwise ORing of two words or bytes:	OR	ORB
• Bitwise XORing of two words or bytes:	XOR	XORB

Compare and Loop Control Instructions

• Comparison of two words or bytes:	CMP	CMPB
• Comparison of two words with post-increment by either 1 or 2:	CMPI1	CMPI2
• Comparison of two words with post-decrement by either 1 or 2:	CMPD1	CMPD2

Boolean Bit Manipulation Instructions

- Manipulation of a maskable bit field
  in either the high or the low byte of a word: BFLDH BFLDL
- Setting a single bit (to ‘1’): BSET
- Clearing a single bit (to ‘0’): BCLR
- Movement of a single bit: BMOV
- Movement of a negated bit: BMOVN
- ANDing of two bits: BAND
- ORing of two bits: BOR
- XORing of two bits: BXOR
- Comparison of two bits: BCMP

Shift and Rotate Instructions

- Shifting right of a word: SHR
- Shifting left of a word: SHL
- Rotating right of a word: ROR
- Rotating left of a word: ROL
- Arithmetic shifting right of a word (sign bit shifting): ASHR

Prioritize Instruction

- Determination of the number of shift cycles required
  to normalize a word operand (floating point support):PRIOR

Data Movement Instructions

- Standard data movement of a word or byte: MOV MOVB
- Data movement of a byte to a word location
  with either sign or zero byte extension: MOVBS MOVBZ

Note: The data movement instructions can be used with a big number of different addressing modes including indirect addressing and automatic pointer in-/decrementing.

System Stack Instructions

- Pushing of a word onto the system stack: PUSH
- Popping of a word from the system stack: POP
- Saving of a word on the system stack,
  and then updating the old word with a new value
  (provided for register bank switching): SCXT

Jump Instructions

- Conditional jumping to an either absolutely, indirectly, or relatively addressed target instruction within the current code segment: JMPA JMPI JMPR
- Unconditional jumping to an absolutely addressed target instruction within any code segment: JMPS
- Conditional jumping to a relatively addressed target instruction within the current code segment depending on the state of a selectable bit: JB JNB
- Conditional jumping to a relatively addressed target instruction within the current code segment depending on the state of a selectable bit with a post-inversion of the tested bit in case of jump taken (semaphore support): JBC JNBS

Call Instructions

- Conditional calling of an either absolutely or indirectly addressed subroutine within the current code segment: CALLA CALLI
- Unconditional calling of a relatively addressed subroutine within the current code segment: CALLR
- Unconditional calling of an absolutely addressed subroutine within any code segment: CALLS
- Unconditional calling of an absolutely addressed subroutine within the current code segment plus an additional pushing of a selectable register onto the system stack: PCALL
- Unconditional branching to the interrupt or trap vector jump table in code segment 0: TRAP

Return Instructions

- Returning from a subroutine within the current code segment: RET
- Returning from a subroutine within any code segment: RETS
- Returning from a subroutine within the current code segment plus an additional popping of a selectable register from the system stack: RETP
- Returning from an interrupt service routine: RETI

System Control Instructions

- Resetting the C164 via software: SRST
- Entering the Idle mode: IDLE
- Entering the Power Down mode: PWRDN
- Servicing the Watchdog Timer: SRVWDT
- Disabling the Watchdog Timer: DISWDT
- Signifying the end of the initialization routine (pulls pin $\overline{\text{RSTOUT}}$ high, and disables the effect of any later execution of a DISWDT instruction): EINIT

Miscellaneous

- Null operation which requires 2 bytes of storage and the minimum time for execution: NOP
- Definition of an unseparable instruction sequence: ATOMIC
- Switch ‘reg’, ‘bitoff’ and ‘bitaddr’ addressing modes to the Extended SFR space: EXTR
- Override the DPP addressing scheme using a specific data page instead of the DPPs, and optionally switch to ESFR space: EXTP EXTPR
- Override the DPP addressing scheme using a specific segment instead of the DPPs, and optionally switch to ESFR space: EXTS EXTSR

Note: The ATOMIC and EXT instructions provide support for uninterruptable code sequences e.g. for semaphore operations. They also support data addressing beyond the limits of the current DPPs (except ATOMIC), which is advantageous for bigger memory models in high level languages. Refer to chapter “System Programming” for examples.*

Protected Instructions

Some instructions of the C164 which are critical for the functionality of the controller are implemented as so-called Protected Instructions. These protected instructions use the maximum instruction format of 32 bits for decoding, while the regular instructions only use a part of it (e.g. the lower 8 bits) with the other bits providing additional information like involved registers. Decoding all 32 bits of a protected doubleword instruction increases the security in cases of data distortion during instruction fetching. Critical operations like a software reset are therefore only executed if the complete instruction is decoded without an error. This enhances the safety and reliability of a microcontroller system.

25 Device Specification

The device specification describes the electrical parameters of the device. It lists DC characteristics like input, output or supply voltages or currents, and AC characteristics like timing characteristics and requirements.

Other than the architecture, the instruction set or the basic functions of the C164 core and its peripherals, these DC and AC characteristics are subject to changes due to device improvements or specific derivatives of the standard device.

Therefore these characteristics are not contained in this manual, but rather provided in a separate Data Sheet, which can be updated more frequently.

Please refer to the current version of the Data Sheet of the respective device for all electrical parameters.

Note: In any case the specific characteristics of a device should be verified, before a new design is started. This ensures that the used information is up to date.

The figure below shows the pin diagram of the C164. It shows the location of the different supply and IO pins. A detailed description of all the pins is also found in the Data Sheet.

Note: Not all alternate functions shown in the figure below are supported by all derivatives.
Please refer to the corresponding descriptions in the data sheets.

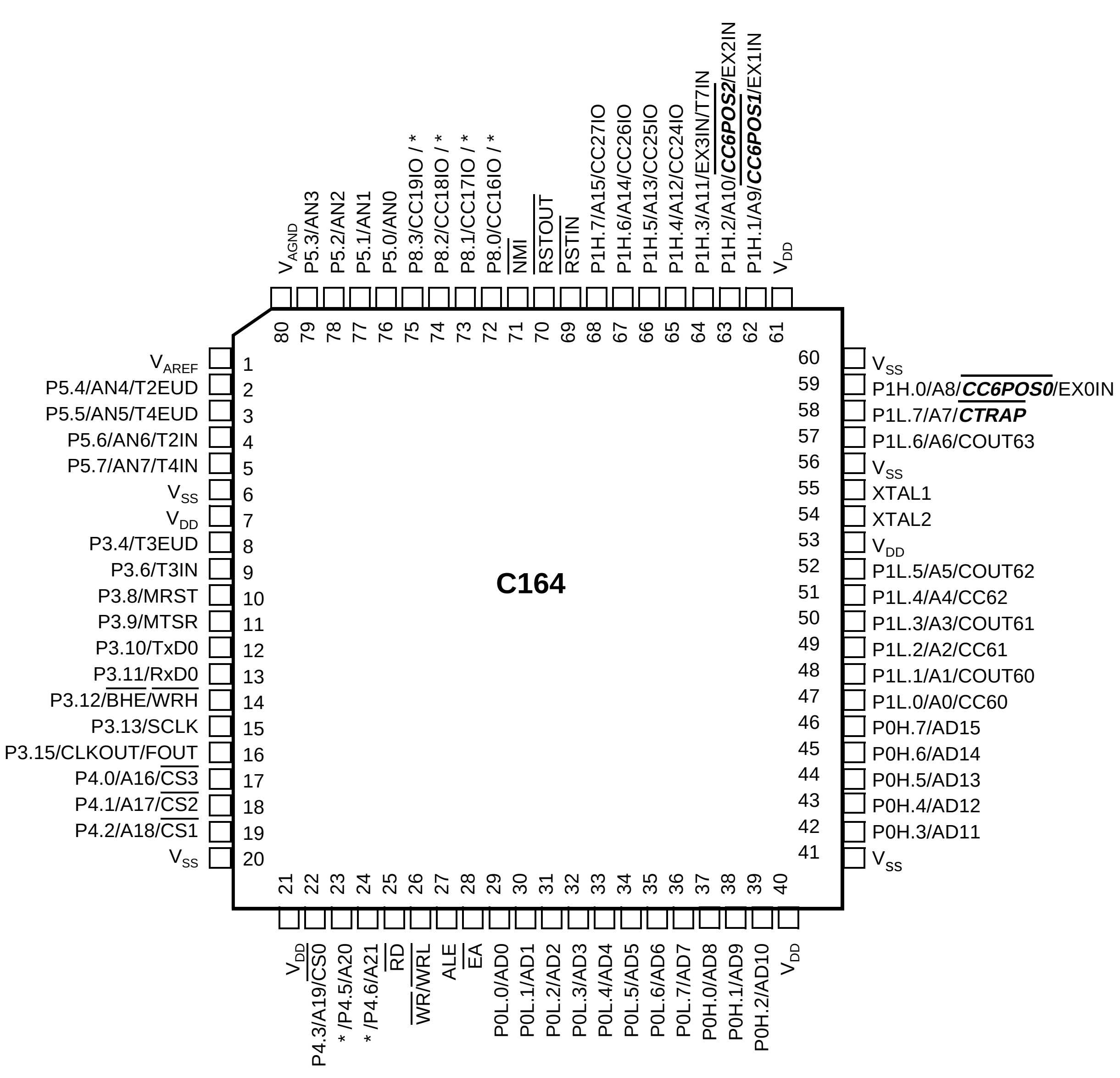


Figure 25-1 Pin Description for C164, P-MQFP-80 Package

***)** The marked pins of Port 4 and Port 8 can have CAN interface lines assigned to them. The CAN module chapter lists the possible assignments.

The ***marked input signals*** are available only in devices with a full function CAPCOM6. They are not available in devices with a reduced CAPCOM6.

26 Keyword Index

This section lists a number of keywords which refer to specific details of the C164 in terms of its architecture, its functional units or functions. This helps to quickly find the answer to specific questions about the C164.

Published by **Infineon Technologies AG**