MMC2107 Device Driver Library

Reference Manual

MMC2107DDLRM/D

MOTOROLA

Freescale Semiconductor, Inc.

Contents

Contents

Section 11 PIT_B Level 1

Section 12 PLL_B Level 1

Section 13 Port_A Level 1

Section 14 QADC64_A Level 1

Section 15 Reset_A Level 1

Contents

List of Figures

List of Tables

Section 1 Introduction

1.1 Overview

The M•CORE™ Device Driver Kit is a software package that accelerates application and higher-level device driver development. The kit provides low-level device drivers for development boards. The kit also includes application code that is immediately operable on the development boards. Kit interface files provide direct register access to all device registers. Additionally, all kit components are fully documented and supported.

1.1.1 Components

- Device Driver API Library
- Device Driver API Interface Files
- Device Driver API Example Applications
- Device Driver Library Reference Manual

1.2 Features

- Reduces application development cycle
- Reduces higher level driver development cycle
- Documented, easy to use, and produces reliable code
- Fully tested and supported
- Contains compilation and link commands provided for supported ANSI C compilers
- Supports a wide variety of M•CORE system peripherals
- Supports multiple M•CORE development system platforms

1.3 Capabilities

- Complete set of functions for low-level device I/O
- Symbolic, direct access to all registers, control fields, and status flags
- Abstraction of I/O addresses and control bit positions
- Configurable runtime parameter checking
- Coded return status
- Integrates with or without an RTOS

1.4 Frequently Asked Questions

1.4.1 What is the M•CORE Device Driver Kit?

The M•CORE Device Driver Kit is a software package that accelerates application and higher-level device driver development. The Kit consists of the Device Driver API Library, the Device Driver API Interface Files, the Device Driver Example Applications, and the Device Driver Library Reference Manual (this document).

The M•CORE Device Driver Kit reduces driver development time by providing example application sources, linker command files, compilation and link utilities, the Device Driver Library itself, and the documentation to utilize these tools. These components are beneficial to programmers who are new to M•CORE products and need to get software running quickly.

1.4.2 Why Use the M•CORE Device Driver API Library?

The M•CORE Device Driver API Library provides symbolic, low-level access to all registers, control fields, and status bits of the device registers. This means that programmers are not required to learn register addresses, or encode control/status bit positions of the hardware programming model. The Device Driver API Library has a simpler, more abstract programming model, resulting in a much faster learning cycle.

As shown in the Device Driver API Example Applications, programs using the Device Driver API Library contain code that is easy to understand and maintain. Cryptic register references and “magic number” constants are eliminated.

The M•CORE Device Driver API Library is compact and low-level, making it well suited for use in higher level driver applications. It can be integrated with or without an RTOS, and with or without interrupts enabled. It contains effective debugging mechanisms such as configurable runtime parameter checking and coded return values. The Library and the Example Applications are fully tested on M•CORE development platforms.

1.4.3 Why Use the M•CORE Device Driver API Example Applications?

M•CORE Device Driver API Example Application source code and build files are provided as learning resources. They can be executed to produce observable peripheral operations on a new development system. The source files contained in this manual provide quick programming references on how to use the API. The sources can also serve as background templates for user modification to explore peripheral operation or to create custom driver applications.

1.4.4 What M•CORE System Peripherals are Supported?

All device drivers for this development system are located after section three of this document. Most on-chip devices such as SCIs, timers, and interrupt controllers are provided. External devices such as flash memory drivers or core devices such as caches and MMUs are provided selectively. Off-board devices such as the DUART and LCD devices on the M•CORE platform board are also provided selectively, but are provided in a separate library.

1.4.5 Can Custom Device Driver Functions be Created?

Yes. You can create custom device drivers in two ways. Custom device drivers can be built from multiple existing functions, from direct register modification, or from combinations of the two. Reasons for creating custom device drivers are to reduce a static set of low level driver calls that do not change during runtime or to compress repetitions of a series of commands.

1.4.6 What is Needed to Start Using the M•CORE Device Driver Kit?

The M•CORE Device Driver Kit provides all build commands necessary to build applications using the M•CORE Device Driver API Library. An M•CORE ANSI C compiler/linker and a download tool are the only other items needed. The compilation and link commands provided with the Example Applications can also be used as references to build new applications.

Section 2 Device Driver Detailed Summary

Note: *Device driver (or module) names are prefaced by the hardware device name or acronym with an underscore followed by a capital letter.*

2.1 Device Driver Kit Components

The Device Driver Kit includes four main components: the Device Driver API Library, the Device Driver API Interface Files, the Device Driver API Example Applications, and the Device Driver Kit Documentation. The location and description of each of these components is described within this section.

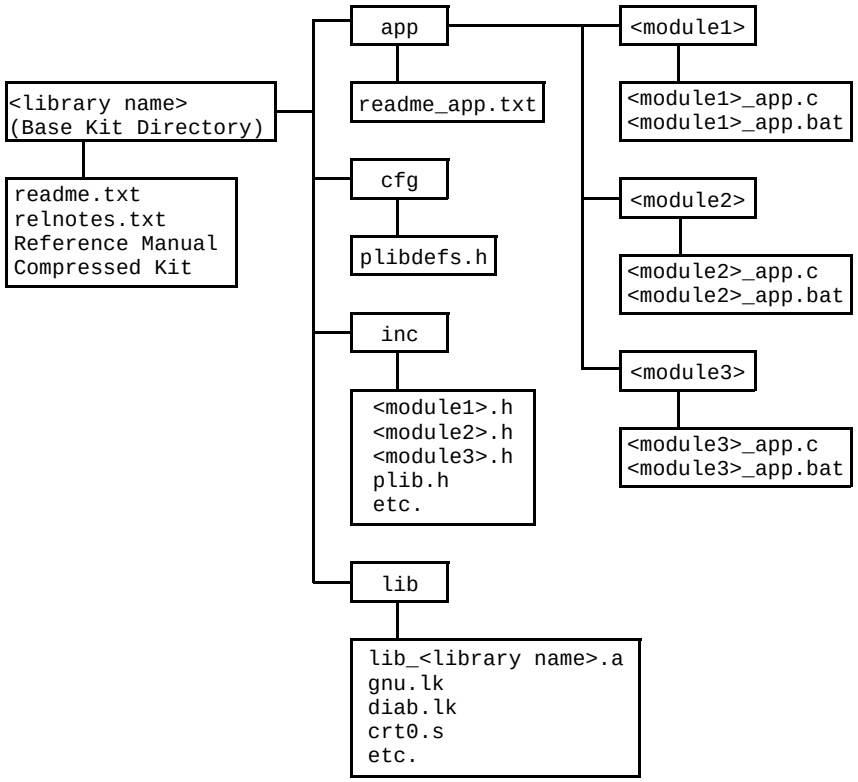

Figure 2-1 Device Driver Kit File Distribution

The above diagram should be read from the left, where the highest level directory of the Device Driver Kit (the library name) exists, to the right, where each directory within the Device Driver Kit is described. Directories are connected to each other horizontally, while the list of files contained within a directory is connected vertically.

2.1.1 Device Driver API Library

LOCATIONS: `<library name>\lib`
FILES: `lib_<library name>.a, gnu.lk, diab.lk, crt0.s`
SUBJECTS: Library characteristics, link files and unique startup files

The library and several related files can be found in the lib directory. The Device Drivers API Library is distributed as a binary object library conforming to the M•CORE Applications Binary Interface (ABI) definition (see M•CORE APPLICATIONS BINARY INTERFACE STANDARDS MANUAL, 1997). It is written in ANSI C, and can be linked into any application that is compiled with a commercially available M•CORE C/C++ compiler that adheres to the ABI standard. For more information on specific features of the library, refer to **2.4 API Features**.

Compiler-specific link files have also been provided in the lib directory. These files are made available for use with the example application source code. Additionally, if a unique startup file (such as crt0.s) is required, it will be in lib.

2.1.2 Device Driver API Interface Files

LOCATIONS: `<library name>\cfg, <library name>\inc`
FILES: `plibdefs.h, <module1>.h, <module2>.h, <module3>.h, plib.h`
SUBJECTS: Modifiable header files, direct register access, global header files

The interface files can be found in both the cfg and the inc directories. The inc directory contains all module-specific header files and several global header files. The cfg directory holds user-modifiable files that contain device register block addresses and driver configuration macros.

Module-specific header files contain register structure definitions, defined mask and bit number values for each register field. These items provide an easy interface for direct register access. An example of direct register access can be found in **2.2.2 Software Relationships**. Also included in the module-specific header files are device-specific type definitions, commonly used data values, function declarations and macros for parameter checking.

Individual device header files are set up to include any higher-level interface files they may require. However, two global header files are of particular interest. The file plib.h is the highest-level peripheral library header file and contains standard type definitions for all modules. The file plibdefs.h is user-modifiable and contains both module register addresses and preprocessor macro definitions for controlling parameter checking.

Note: *Only files contained in the CFG subdirectory are user-modifiable. Changing other header files may adversely affect library operations.*

2.1.3 Device Driver API Example Applications

LOCATIONS:	`<library name>\app, <library name>\app\<module1>, <library name>\app\<module2>, <library name>\app\<module3>, etc.`
FILES:	`readme_plib.h, <module1>_app.bat, <module1>_app.c, <module2>_app.bat, <module2>_app.c, <module3>_app.bat, <module3>_app.c, etc.`
SUBJECTS:	Application setup information, building and running applications

Example application code can be found in the app directory and its subdirectories. The app directory contains readme_app.txt and a subdirectory for each module contained in the library. The readme_app.txt file contains general setup information required to run all the example applications. This includes several compiler and debugger configurations, hardware components, and general steps to build and run the application. Refer to the specific application source file to determine if there are any device-specific configurations, components or setup needed.

Each of the module subdirectories contain a module-specific build file and one or more module-specific example application source code files. The build file contains one or more sets of commands for a specific compiler to build the application. Edit this file to comment out all compiler commands except the compiler of choice. The source file contains both source code and a short description of the application's operation. The source file also contains any unique setup requirements beyond those covered in readme_app.txt. The application is organized to show the user many of the driver function calls in a way that uses the device features.

2.1.4 Device Driver Kit Documentation

LOCATIONS:	`<library name>`
FILES:	`readme_plib.txt, relnotes.txt, Reference Manual, <module2>_app.bat, <module2>_app.c, <module3>_app.bat, <module3>_app.c, etc.`
SUBJECTS:	Reference Manual, release notes, compressed Device Driver Kit

The documentation can be found in the base kit directory. This directory contains readme_plib.txt, relnotes.txt, and the Device Driver Library Reference Manual. A compressed copy of the Device Driver's Kit is contained on the CD shipped with the Development System.

2.2 System Relationship

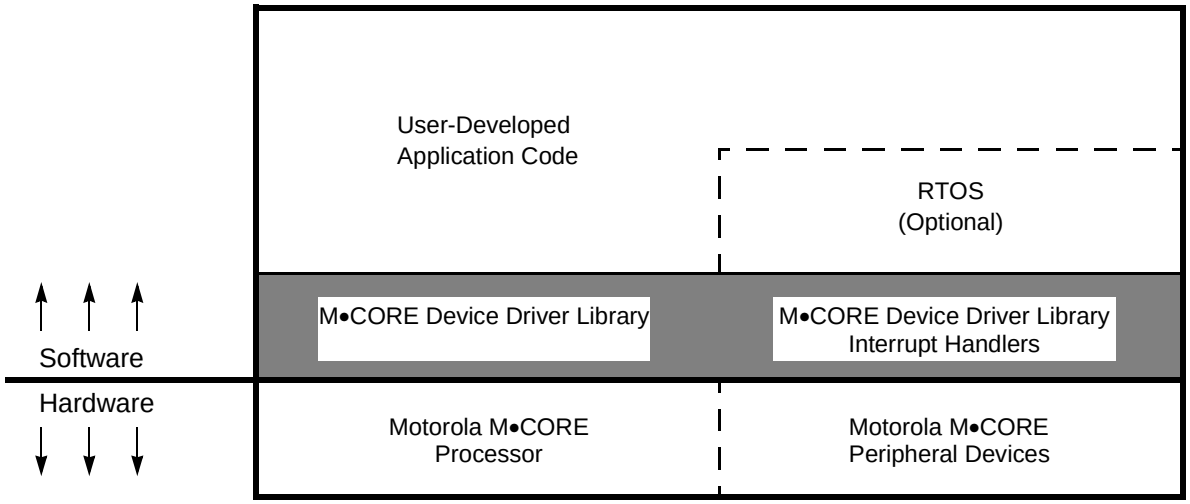

Figure 2-2 Device Driver Library System Relationships

2.2.1 Hardware Relationships

2.2.1.1 M•CORE

Selected core-based hardware devices have an associated driver. These drivers access the hardware device registers or the associated control mechanisms to control the device.

2.2.1.1.1 On-Chip Devices

Unless otherwise noted, each on-chip hardware device has an associated driver. These drivers access the hardware device registers or the associated control mechanisms to control the device.

2.2.1.1.2 External Devices

External hardware devices such as those contained on the M•CORE Platform Board (MPFB1200) have associated drivers in a separate library. These drivers access the hardware device registers or the associated control mechanisms to control the device through the memory mapping of the target processor. External memory controllers, such as flash drivers, are available selectively. More information on these libraries can be found at the M•CORE Device Drivers web site (http://mcore.sps.mot.com/software/sw_drivers.html).

2.2.2 Software Relationships

2.2.2.1 Direct Register Access

All Device Driver API modules contain generic GetRegister and SetRegister routines for arbitrarily manipulating device registers. They are all written in a uniform way to make them easy to use between and among modules. However, one can access registers in an even more direct way without departing from C language syntax. The handle provided for any Level 1 library routine refers to the base address of the memory-mapped device register block. All device register blocks are mapped to a C language structure as defined in the header file for the appropriate module. Therefore, the handle may be used as a structure pointer to access device registers directly.

For example, below is the register definition of a pulse width modulator (PWM):

```
typedef struct
        {
        volatile u2  PWMCR;     /* Control Register */
        u2  PWMPR;              /* Period Register  */
        u2  PWMWR;              /* Width Register   */
        volatile u2  PWMCTR;    /* Counter Register */
        }
PWM_A_t, *pPWM_A_t;
```

NOTE: *On certain modules' register definitions, blocks of memory may be noted as reserved. This space is necessary to preserve the form of the register definition. Do not access these blocks.*

To obtain a handle to the first PWM channel, the following syntax may be used:

```
pPWM_A_t pwm0 = (pPWM_A_t)__MMC2001_PWM0;
```

The identifier __MMC2001_PWM0 is defined in the user-modifiable plibdefs.h file, and instantiates the address of the initial PWM channel as a manifest constant. The declaration above allocates storage for a pointer to a structure of type PWM_A_t, which reflects the definition of the PWM device registers. The pwm0 variable provides direct access to the PWM registers for this channel.

The following assignments would suffice to set the PWM channel 0 period and width registers:

```
#define 50_PERCENT_DUTY_CYCLE                0x200
#define 50_PERCENT_WIDTH                     0x100

pwm0->PWMPR = 50_PERCENT_DUTY_CYCLE;         /* set for 50% duty cycle */
pwm0->PWMWR = 50_PERCENT_WIDTH;              /* set for 50% width */
```

The code below can be used to enable the PWM and run with a clock divider of 256:

```
pwm0->PWMCR |= PWMCR_COUNTEN_MASK | PWMCR_CLKSEL_MASK;
```

The following code disables the PWM channel 0 counter:

```
pwm0->PWMCR &= ~ PWMCR_COUNTEN_MASK;
```

2.2.2.2 Library Functions

The Device Driver API functions are designed to allow the programmer to control the device at a near-register level, but without the concern of exact positions of registers and fields. The functions may also allow device control without knowledge of the detailed sequences of register operations. Some contain individual operations such as transmitting and receiving data. Others are collected into logical groups such as initialization operations or controlling several related operations simultaneously such as IrDA and loopback. Although many of the device operations have been mapped to API functions, some may not be available.

For example, below is a possible API function call for a pulse width modulator (PWM) to force a new period with a 50% duty cycle and a 50% width:

```
#define 50_PERCENT_DUTY_CYCLE                0x200
#define 50_PERCENT_WIDTH                     0x100
pPWM_A_t pwm0 = (pPWM_A_t)__MMC2001_PWM0;
ddErr_t retval;
...

/* Call PWM_A_UpdateOutput */
retval = PWM_A_UpdateOutput (pwm0,                /* PWM0 base address */
                             ddFALSE,             /* force new period */
                             50_PERCENT_DUTY_CYCLE, /* 50% duty cycle */
                             50_PERCENT_WIDTH);   /* 50% width */
```

2.2.2.3 Interrupt Integration

Device Driver Library routines do not explicitly deal with device interrupts. The API of a service at this level presents the programmer with a model of the device essentially at the hardware level, although it provides the programmer with symbolic device access and hides the details of register field layout and manipulation sequences. The programmer can choose to utilize a peripheral service exclusively at this level. Drivers do not process interrupts but interrupt servicing can be developed in conjunction to an interrupt controller driver (if available)

2.2.2.4 RTOS Integration

There are no RTOS dependencies within any driver functions. However, the API does not exclude the user from RTOS or kernel calls to control driver operation. Contact the RTOS vendor to determine if this library has been incorporated in a board support package for your development board.

2.3 Service Levels

The device driver module for a given device may be viewed as a service in support of that device. Device services can be categorized by the degree of device abstraction that the service presents and the amount of interrupt processing support it provides. Library functions access device information through a device handle, which is typically (but not exclusively) the first parameter of each function.

2.3.1 Level 1 Services

Level 1 services are at the lowest level; they interact directly with the hardware and return immediately to the caller, unless otherwise noted. The device handle in a level 1 call is always the base address of the device register block.

Being essentially at the level of the raw hardware, a level 1 service has minimal interrupt support, although a higher level service employing interrupts may be built upon a level 1 service. The main benefit of a level 1 service is that it is symbolic; there are no hardware register names or structures to remember, and the useful capability of parameter checking is also available. All M•CORE peripheral library modules provide an interface to level 1 services.

2.3.2 Level 2 Services

A level 2 service presents the user with a more abstract model of a given device than does a level 1 service. It is generally built upon a level 1 service, although it may make use of optimizations unavailable to the application programmer. An application program that uses a level 2 service should call only level 2 functions (although some of these may be simple passthroughs to lower-level functions).

2.4 API Features

2.4.1 Naming Conventions

The naming conventions for both the libraries and the function names have been established to provide an intuitive and simple reference for the users. Each library name follows the formula of <processor>_<maj_rev>_<min_rev>. The <processor> specifies the processor associated with this library, but in unique situations, it may be the core name or the name of a support board (such as the M•CORE Platform Board). The <maj_rev> and <min_rev> are the major revision number and minor revision numbers, respectively. Revisions are incremented by a whole number for major changes to the library, by tenths of a number for minor changes to the library, and by a character for a non-library change to the kit. Therefore, the name of the MMC2001 processor library that is on its initial major revision (1), but has had both a minor change (1) and a non-library change (b) would be would be mmc2001_1_1b.

Function names are intended to convey sufficient understanding of the operation of the function. Each function name attempts to follow the formula of <module>_<functional descriptor>, where <module> corresponds to the module acronym and version. The <functional descriptor> is of the form of a verb-noun phrase. The verb is for the following list (whenever possible) when the verb description matches the function’s operation. The noun either matches the register or register field name being operated on, the operational intent, or a combination of the two. Therefore, the name of a UART_A function that controls the baud rate might be UART_A_ControlBaudRate, and a function that provides initialization to the FSC_A module might be called FSC_A_Init.

The following is a list of some verbs and the description of their usage:

Table 2-1 Verb Usage

Verb	Description
Init	This function initializes all parameters in a module, either to a user-defined state or to their reset values. It is used to initialize, re-initialize, or reset the state of the driver.
Control	This function writes a register or register field based on user-specified parameters. It is used to provide a sustained effect on the operation of the driver.
Transmit	This function writes a register or register field based on user-specified parameters. It is used to send data to another device external to the driver.
Receive	This function reads a register or register field to a user-specified location. It is used to receive data from another device external to the driver.
Reset	This function writes a register or register field to its reset value, which may or may not be logic zero. It is used to reset the state of the register or register field.
Clear	This function writes a register or register field to logic zero. It is used to clear the register or register field.
Set	This function writes a register or register field based on user-specified parameters. It is used to write the state of the register or register field with little or no formatting.
Get	This function reads a register or register field to a user-specified location. It is used to read the state of the register or register field with little or no formatting.

2.4.2 Error Codes

There are currently two types of error code conventions. The legacy convention was established with early libraries. Because of maintenance difficulties and concerns in using multiple M•CORE Device Driver Kits, our current convention was created to correct these problems. Regardless of the convention, multiple M•CORE Device Driver API Libraries are compatible.

2.4.2.1 Legacy Convention

All device driver library functions return a status of type ddErr_t, an enumeration defined in the global errors.h header file. If a function encounters no error conditions during its operation, it will return a status of DD_ERR_NONE. In most cases a returned status other than DD_ERR_NONE indicates some type of error situation. An exception to this rule would be when a function returned information concerning an out-of-band condition such as a received break signal on a serial line.

Return codes are grouped by module type, so that the actual enumeration values for a given module are sequential. Return code symbols are prefixed by the module name. There is a small group of global return codes with the prefix DD_. Do not make modifications to this file, as serious damage to the library could result when reporting error codes.

2.4.2.2 Current Convention

All device driver library functions return a driver-specific status defined in the device-specific header file. If a function encounters no exceptional conditions during its operation, it will return a status of <module>_ERR_NONE where <module> is the device-specific module name. In most cases a returned status other than <module>_ERR_NONE indicates some type of error situation. An exception to this rule would be when a function returned information concerning an out-of-band condition such as a received break signal on a serial line. All error codes that are unique only in their module name prefix are considered equivalent. Therefore, assume the UART_B_ERR_INVALID_ADDRESS and PWM_C_ERR_INVALID_ADDRESS pertain to an invalid address specified in both a UART_B and a PWM_C function.

2.4.3 Library Macros

All library macros may be found within the device-specific header file of the module.

2.4.3.1 Parameter Checking Macro

The peripheral library API definitions are actually made in terms of C language preprocessor macros. This was done to efficiently implement selective API parameter checking. Enabling parameter checking during early development is recommended to assist in the debugging of parameters, but disable parameter checking after time to reduce code size. The convention for naming addressable API functions is to append an underscore-lowercase-F (_f) to the API name. For example, the UART initialization API is called UART_A_Init, but the addressable function corresponding to this API definition is called UART_A_Init_f. This knowledge is useful when attempting to load API function addresses into a table. However, in most cases using the standard API definition is appropriate.

Parameter checking code is introduced at application compile time by setting the <module>_PARAM_CHECKING flag (defined initially in plibdefs.h) to a non-zero value, where <module> is the name of the peripheral device in question. Here is an example:

```
#undef  SIM_A_PARAM_CHECKING
#define SIM_A_PARAM_CHECKING 1       /* Turn on SIM param checking */

/* Perform parameter checking on initialization */
if (SIM_A_ClearTODAlarmInterrupt(SIMPtr) != DD_ERR_NONE)
    ...

#undef  SIM_A_PARAM_CHECKING
#define SIM_A_PARAM_CHECKING 0        /* Turn off SIM param checking */

/* No parameter checking on enable */
if (SIM_A_ServiceWD(SIMPtr) != DD_ERR_NONE)
    ...
```

Note that the PARAM_CHECKING flag may be used selectively, thereby activating parameter checking in only certain API invocations.

2.4.3.2 Default Macros

Default macros generally have the same name as the documented API function, except that all characters in the macro name are uppercase and _DEFAULT is appended to the end. The only macro parameter required is the device descriptor parameter. Here is an example:

```
/*****************************************************************/
/* Macro:       UART_A_INIT_DEFAULT                              */
/*                                                               */
/* Purpose:     Call UART_A_Init with default parameters.        */
/*                Make a copy of this macro to customize defaults.*/
/*                                                               */
/*****************************************************************/
#define UART_A_INIT_DEFAULT(UARTPtr)                             \
    UART_A_Init(                                                 \
                UARTPtr,                                         \
                UART_A_DEFAULT_DIVIDER,                          \
                UART_A_DEFAULT_SIZE,                             \
                UART_A_DEFAULT_PARITY,                           \
                UART_A_DEFAULT_STOP_BITS,                        \
                UART_A_DEFAULT_RX_TRIG,                          \
                UART_A_DEFAULT_TX_TRIG,                          \
                UART_A_DEFAULT_RTS_INT,                          \
                UART_A_DEFAULT_DOZE,                             \
                UART_A_DEFAULT_FLOW,                             \
                UART_A_DEFAULT_UART_PINS,                        \
                UART_A_DEFAULT_OUTPUT_PINS                       \
               )
```

The default values for parameters such as word size, parity, and number of stop bits are supplied in the macro definition. These default values are defined in the corresponding module header file.

2.4.4 Reentrancy and Blocking

The M•CORE Device Driver API functions are reentrant with respect to non-peripheral registers (e.g. core processor registers). The functions contain no state information and no global variables are required. This allows the functions to remain independent of board-dependent information such as specific device addresses.

The functions are also non-blocking, unless otherwise noted. Any blocking will be specified in an API function's description. Cases in which blocking may be used include status-dependent operations that may cause device failure if the driver state is not reset. For example, initiating another command on the flash before the flash driver completes an erase could cause the flash module to become inoperable.

2.5 Contact Information

Updates and future revisions of this library can be found at the M•CORE Device Drivers web page (http://mcore.sps.mot.com/software/index.html). The M•CORE Device Drivers team may also be contacted by email (dev_drivers_help@lakewood.sps.mot.com). Problems should be submitted by email to this address.

Section 3 Library Specific Information

3.1 Library Name

The library described in this document is MMC2107.

3.2 Application Supported Compilers

The example application build files support:

- CodeWarrior for M•CORE™ Embedded Systems, rev 1.3

3.3 Core Specific Definitions

None.

3.4 Microprocessor and Board Specific Definitions

Table 3-1 Board Specific Definitions

Identifier	Value
__MMC2107_CMFR_FLASH	0x00000000
__MMC2107_SRAM	0x00800000
__MMC2107_PORTS	0x00C00000
__MMC2107_CCM	0x00C10000
__MMC2107_CS	0x00C20000
__MMC2107_PLL	0x00C30000
__MMC2107_RESET	0x00C40000
__MMC2107_ITCN	0x00C50000
__MMC2107_EDGEPORT	0x00C60000
__MMC2107_WATCHDOG	0x00C70000
__MMC2107_PIT1	0x00C80000
__MMC2107_PIT2	0x00C90000
__MMC2107_QADC64	0x00CA0000
__MMC2107_SPI	0x00CB0000
__MMC2107_SCI1	0x00CC0000
__MMC2107_SCI2	0x00CD0000
__MMC2107_CMFR_CTL	0x00D00000
__MMC2107_AMD_FLASH	0x80000000
__MMC2107_MMIO	0x817FFFFD
__MMC2107_CS0	0x80000000
__MMC2107_CS1	0x80800000
__MMC2107_CS2	0x81000000

Table 3-1 Board Specific Definitions

Identifier	Value
__MMC2107_CS3	0x81800000
__PFB1200_DUART_EXT_CLK	1843200
__PFB1200_DUART_CHANNEL_A_OFFSET	0x00000020
__PFB1200_DUART_CHANNEL_B_OFFSET	0x00000000
__PFB1200_LCD_OFFSET	0x00000080
__PFB1200_EXT_LCD_OFFSET	0x000000C0

3.5 Special Notes or Exceptions

The error code convention used by this library is the current convention. See section **2.4.2.2Current Convention** for more details

Please refer to the file relnotes_mmc2107.txt for the hardware limitations of the software device drivers

3.6 References

Refer to the following documents for more information:

- MMC2107 Advance Information (Data Book) (MMC2107/D)
- Motorola M•CORE Reference Manual (MCORERM/AD)
- Motorola M•CORE Applications Binary Interface Standards Manual (MCOREABI/D)

Section 4 AMD_FLASH_B Level 1

This section describes the uses for the M•CORE™ Device Driver AMD_FLASH_B Level 1.

4.1 Overview

The Advanced Micro Devices Flash (AMD_FLASH) and its accompanying device driver (AMD_FLASH_B) are used on this development system. The Level 1 service is designed to allow the programmer to control the AMD_FLASH on a near-register level, but without the concern of exact positions of registers and fields, or the detailed sequences of register operations. For more information on the AMD_FLASH module, refer to the AMD AM29LV800B Flash Specification available on the external web at http://www.mot.com/SPS/MCORE/tools_drivers.html.

4.2 AMD_FLASH_B Service Functions

The AMD_FLASH_B device driver software provides these functions:

- Erases a sector or the entire flash based on user input
- Suspends any sector erase operation
- Resumes any suspended sector erase
- Implements the flash write algorithm to write data to the flash
- Enables/disables the unlock bypass mode
- Gets the manufacturer ID
- Gets the device ID
- Returns whether a given sector is or is not protected
- Determines whether the data in the address specified by the user is blank
- Determines whether the data in a user-specified address is written correctly

4.3 AMD_FLASH_B Data Type Definitions

The following paragraphs show standard, board specific, abstract and enumerated data type definitions.

4.3.1 Standard Data Type Definitions

- UINT8 — 8-bit unsigned data value
- INT8 — 8-bit signed data value
- UINT16 — 16-bit unsigned data value
- INT16 — 16-bit signed data value

- UINT32 — 32-bit unsigned data value
- INT32 — 32-bit signed data value
- AMD_FLASH_B_Boolean_t — boolean data value (32 bits)

4.3.2 Board Specific Base Address Definitions

```
#define __M340_AMD_FLASH_B (pAMD_FLASH_B_t(0x00000000))
```

4.3.3 Abstract Data Type Definitions

4.3.3.1 AMD_FLASH_B_t – AMD Flash Register Definition

```
typedef UINT32 pAMD_FLASH_B_t;      /* defines the base address */
```

4.3.4 Enumerated Data Type Definitions

4.3.4.1 AMD_FLASH_B_SECTOR_t – AMD FLASH Mode Selection

```
typedef enum {
                  AMD_FLASH_B_ALL,           /* entire chip */
                  AMD_FLASH_B_SA0,           /* sector SA0 */
                  AMD_FLASH_B_SA1,           /* sector SA1 */
                  AMD_FLASH_B_SA2,           /* sector SA2 */
                  AMD_FLASH_B_SA3,           /* sector SA3 */
                  AMD_FLASH_B_SA4,           /* sector SA4 */
                  AMD_FLASH_B_SA5,           /* sector SA5 */
                  AMD_FLASH_B_SA6,           /* sector SA6 */
                  AMD_FLASH_B_SA7,           /* sector SA7 */
                  AMD_FLASH_B_SA8,           /* sector SA8 */
                  AMD_FLASH_B_SA9,           /* sector SA9 */
                  AMD_FLASH_B_SA10,          /* sector SA10 */
                  AMD_FLASH_B_SA11,          /* sector SA11 */
                  AMD_FLASH_B_SA12,          /* sector SA12 */
                  AMD_FLASH_B_SA13,          /* sector SA13 */
                  AMD_FLASH_B_SA14,          /* sector SA14 */
                  AMD_FLASH_B_SA15,          /* sector SA15 */
                  AMD_FLASH_B_SA16,          /* sector SA16 */
                  AMD_FLASH_B_SA17,          /* sector SA17 */
                  AMD_FLASH_B_SA18           /* sector SA18 */
} AMD_FLASH_B_SECTOR_t;
```

4.3.5 Register Macros

4.3.5.1 Sector Addresses – AMD Flash Sector Selection

```
/# for Am29LV800BB chip bottom boot block sector #/
#define AMD_FLASH_B_SA0_BOTTOM_BOOT 0x00000
#define AMD_FLASH_B_SA1_BOTTOM_BOOT 0x02000
#define AMD_FLASH_B_SA2_BOTTOM_BOOT 0x03000
```

```
#define AMD_FLASH_B_SA3_BOTTOM_BOOT 0x04000
#define AMD_FLASH_B_SA4_BOTTOM_BOOT 0x08000
#define AMD_FLASH_B_SA5_BOTTOM_BOOT 0x10000
#define AMD_FLASH_B_SA6_BOTTOM_BOOT 0x18000
#define AMD_FLASH_B_SA7_BOTTOM_BOOT 0x20000
#define AMD_FLASH_B_SA8_BOTTOM_BOOT 0x28000
#define AMD_FLASH_B_SA9_BOTTOM_BOOT 0x30000
#define AMD_FLASH_B_SA10_BOTTOM_BOOT 0x38000
#define AMD_FLASH_B_SA11_BOTTOM_BOOT 0x40000
#define AMD_FLASH_B_SA12_BOTTOM_BOOT 0x48000
#define AMD_FLASH_B_SA13_BOTTOM_BOOT 0x50000
#define AMD_FLASH_B_SA14_BOTTOM_BOOT 0x58000
#define AMD_FLASH_B_SA15_BOTTOM_BOOT 0x60000
#define AMD_FLASH_B_SA16_BOTTOM_BOOT 0x68000
#define AMD_FLASH_B_SA17_BOTTOM_BOOT 0x70000
#define AMD_FLASH_B_SA18_BOTTOM_BOOT 0x78000

/# for Am29LV800BT chip Top Boot Block sector #/
#define AMD_FLASH_B_SA0_TOP_BOOT 0x00000
#define AMD_FLASH_B_SA1_TOP_BOOT 0x08000
#define AMD_FLASH_B_SA2_TOP_BOOT 0x10000
#define AMD_FLASH_B_SA3_TOP_BOOT 0x18000
#define AMD_FLASH_B_SA4_TOP_BOOT 0x20000
#define AMD_FLASH_B_SA5_TOP_BOOT 0x28000
#define AMD_FLASH_B_SA6_TOP_BOOT 0x30000
#define AMD_FLASH_B_SA7_TOP_BOOT 0x38000
#define AMD_FLASH_B_SA8_TOP_BOOT 0x40000
#define AMD_FLASH_B_SA9_TOP_BOOT 0x48000
#define AMD_FLASH_B_SA10_TOP_BOOT 0x50000
#define AMD_FLASH_B_SA11_TOP_BOOT 0x58000
#define AMD_FLASH_B_SA12_TOP_BOOT 0x60000
#define AMD_FLASH_B_SA13_TOP_BOOT 0x68000
#define AMD_FLASH_B_SA14_TOP_BOOT 0x70000
#define AMD_FLASH_B_SA15_TOP_BOOT 0x78000
#define AMD_FLASH_B_SA16_TOP_BOOT 0x7C000
#define AMD_FLASH_B_SA17_TOP_BOOT 0x7D000
#define AMD_FLASH_B_SA18_TOP_BOOT 0x7E000
```

4.3.5.2 Miscellaneous

```
/* Command Sequence Address offsets */
#define COMMAND_ADDRESS_1           0x2AA    /* AMD FLASH 0x2AA offset */
#define COMMAND_ADDRESS_2           0x555    /* AMD FLASH 0x555 offset */

/* AMD_FLASH_B Miscellaneous Defines */
#define TOP_BOOT_BLOCK_WORD         0x22DA
#define BOTTOM_BOOT_BLOCK_WORD      0x225B
#define ERASED_WORD                 0xFFFFFFFF
#define SECTOR_PROTECT_ON_WORD      0x010001

/* Hard coded AMD Flash commands */
#define AMD_FLASH_B_COMMAND_00      0x00000000
#define AMD_FLASH_B_COMMAND_10      0x00100010
```

```
#define AMD_FLASH_B_COMMAND_20      0x00200020
#define AMD_FLASH_B_COMMAND_30      0x00300030
#define AMD_FLASH_B_COMMAND_55      0x00550055
#define AMD_FLASH_B_COMMAND_80      0x00800080
#define AMD_FLASH_B_COMMAND_90      0x00900090
#define AMD_FLASH_B_COMMAND_A0      0x00A000A0
#define AMD_FLASH_B_COMMAND_AA      0x00AA00AA
#define AMD_FLASH_B_COMMAND_B0      0x00B000B0
#define AMD_FLASH_B_COMMAND_F0      0x00F000F0
```

4.3.6 Return Codes

Return codes from the enumerated type “AMD_FLASH_B_ReturnCode_t”.

Return Code	Functions Returning	Description
AMD_FLASH_B_ERR_NONE	All	No error.
AMD_FLASH_B_ERR_BAD_RETURN_ADDR	GetDeviceID, GetManufacturerID, VerifySectorProtect	Result pointer is zero.
AMD_FLASH_B_ERR_INVALID_SECTOR	Erase, VerifySectorProtect	Sector not valid.

4.4 AMD_FLASH_B API Definitions

4.4.1 AMD_FLASH_B_Erase

4.4.1.1 Description

The AMD_FLASH_B_Erase function erases a sector or the entire flash chip based on user input. This function writes to the flash command addresses 2 and 1. A chip erase typically takes 14 seconds and a sector erase typically takes 0.7 seconds, with a worst case of 15 seconds. Parameter checking is a compile-time option.

4.4.1.2 Prototype

```
AMD_FLASH_B_ReturnCode_t AMD_FLASH_B_Erase(
                  pAMD_FLASH_B_t              AMD_FLASHPtr,
                  AMD_FLASH_B_SECTOR_t        Sector
)
```

4.4.1.3 Arguments

Data Type	Formal Name	Description	Valid Values
pAMD_FLASH_B_t	AMD_FLASHPtr	AMD_FLASH_B base address associated with this driver.	Any non-zero core processor data address.
AMD_FLASH_B_SECTOR_t	Sector	Erase performed on the entire chip (_ALL) or an individual sector.	AMD_FLASH_B_ALL, AMD_FLASH_B_SA0 Through AMD_FLASH_B_SA18

4.4.1.4 Return Values

Error Code	Description
AMD_FLASH_B_ERR_NONE	No error
AMD_FLASH_B_ERR_INVALID_SECTOR	Sector given not valid.

4.4.2 AMD_FLASH_B_SuspendErase

4.4.2.1 Description

AMD_FLASH_B_SuspendErase suspends any erase operation currently executing. The erase suspend is valid only during a sector erase operation. The system may read and program in non-erasing sectors, or enter the autoselect mode, when in the erase suspend mode. The only way to resume the erase is to call AMD_FLASH_B_ResumeErase. Parameter checking is a compile-time option.

4.4.2.2 Prototype

```
AMD_FLASH_B_ReturnCode_t AMD_FLASH_B_SuspendErase(
                                    pAMD_FLASH_B_t AMD_FLASHPtr,
                                     )
```

4.4.2.3 Arguments

Data Type	Formal Name	Description	Valid Values
pAMD_FLASH_B_t	AMD_FLASHPtr	AMD_FLASH_B base address associated with this driver.	Any non-zero core processor data address.

4.4.2.4 Return Values

Error Code	Description
AMD_FLASH_B_ERR_NONE	No error

4.4.3 AMD_FLASH_B_ResumeErase

4.4.3.1 Description

AMD_FLASH_B_ResumeErase resumes any suspended erase command. A multi-threaded program is needed to make use of this function. This function is valid only during the erase suspend. Parameter checking is a compile-time option.

4.4.3.2 Prototype

```
AMD_FLASH_B_ReturnCode_t AMD_FLASH_B_ResumeErase(
                           pAMD_FLASH_B_t    AMD_FLASHPtr,
                           )
```

4.4.3.3 Arguments

Data Type	Formal Name	Description	Valid Values
pAMD_FLASH_B_t	AMD_FLASHPtr	AMD_FLASH_B base address associated with this driver.	Any non-zero core processor data address.

4.4.3.4 Return Values

Error Code	Description
AMD_FLASH_B_ERR_NONE	No error

4.4.4 AMD_FLASH_B_Program

4.4.4.1 Description

AMD_FLASH_B_Program executes the normal flash program algorithm to write user-specified data to a 32-bit flash address. This function assumes the target address has been erased. This function writes to the flash command addresses 2 and 1. Parameter checking is a compile-time option.

4.4.4.2 Prototype

```
AMD_FLASH_B_ReturnCode_t AMD_FLASH_B_Program(
                           pAMD_FLASH_B_t    AMD_FLASHPtr,
                           UINT32            * TargetPtr,
                           UINT32            Word
                           )
```

4.4.4.3 Arguments

Data Type	Formal Name	Description	Valid Values
pAMD_FLASH_B_t	AMD_FLASHPtr	AMD_FLASH_B base address associated with this driver.	Any non-zero core processor data address.
UINT32 *	TargetPtr	Target address where word is to be written	Any non-zero flash address.
UINT32	Word	Data to be written	Any 32-bit value.

4.4.4.4 Return Values

Error Code	Description
AMD_FLASH_B_ERR_NONE	No error

4.4.5 AMD_FLASH_B_ControlUnlockBypass

4.4.5.1 Description

AMD_FLASH_B_ControlUnlockBypass enables or disables the unlock bypass mode. This function writes to the flash command addresses 2 and 1. Parameter checking is a compile-time option.

4.4.5.2 Prototype

```
AMD_FLASH_B_ReturnCode_t AMD_FLASH_B_ControlUnlockBypass(
                           pAMD_FLASH_B_t    AMD_FLASHPtr,
                           AMD_FLASH_B_Boolean_t EnableBypass
                                    )
```

4.4.5.3 Arguments

Data Type	Formal Name	Description	Valid Values
pAMD_FLASH_B_t	AMD_FLASHPtr	AMD_FLASH_B base address associated with this driver.	Any non-zero core processor data address.
AMD_FLASH_B_Boolean_t	EnableBypass	Enable or disable the bypass programming mode	AMD_FLASH_B_TRUE = Enable bypass mode AMD_FLASH_B_FALSE = Disable bypass mode

4.4.5.4 Return Values

Error Code	Description
AMD_FLASH_B_ERR_NONE	No error

4.4.6 AMD_FLASH_B_ProgramUnlockBypass

4.4.6.1 Description

AMD_FLASH_B_ProgramUnlockBypass executes the unlock bypass flash program algorithm to write user-specified data to a 32-bit flash address. This function writes to the flash command address 1. Parameter checking is a compile-time option.

4.4.6.2 Prototype

```
AMD_FLASH_B_ReturnCode_t AMD_FLASH_B_ProgramUnlockBypass (
                                    pAMD_FLASH_B_t AMD_FLASHPtr,
                                    UINT32   * TargetPtr,
                                    UINT32   Word
                                     )
```

4.4.6.3 Arguments

Data Type	Formal Name	Description	Valid Values
pAMD_FLASH_B_t	AMD_FLASHPtr	AMD_FLASH_B base address associated with this driver.	Any non-zero core processor data address.
UINT32 *	TargetPtr	Target address where word is to be written	Any non-zero flash address.
UINT32	Word	Data to be written	Any 32-bit value.

4.4.6.4 Return Values

Error Code	Description
AMD_FLASH_B_ERR_NONE	No error

4.4.7 AMD_FLASH_B_GetManufacturerID

4.4.7.1 Description

AMD_FLASH_B_GetManufacturerID gets the manufacturer ID. Parameter checking is a compile-time option.

4.4.7.2 Prototype

```
AMD_FLASH_B_ReturnCode_t AMD_FLASH_B_GetManufacturerID (
                                    pAMD_FLASH_B_t AMD_FLASHPtr,
                                    UINT32   * ResultPtr
                                     )
```

4.4.7.3 Arguments

Data Type	Formal Name	Description	Valid Values
pAMD_FLASH_B_t	AMD_FLASHPtr	AMD_FLASH_B base address associated with this driver.	Any non-zero core processor data address.
UINT32 *	ResultPtr	Result address in which to write manufacturer ID	Any non-zero core processor data address.

4.4.7.4 Return Values

Error Code	Description
AMD_FLASH_B_ERR_NONE	No error
AMD_FLASH_B_ERR_BAD_RESULT_ADDR	Result address is NULL

4.4.8 AMD_FLASH_B_GetDeviceID

4.4.8.1 Description

AMD_FLASH_B_GetDeviceID gets the device ID. This function writes to the flash command addresses 2 and 1. Parameter checking is a compile-time option.

4.4.8.2 Prototype

```
AMD_FLASH_B_ReturnCode_t AMD_FLASH_B_GetDeviceID(
                           pAMD_FLASH_B_t    AMD_FLASHPtr,
                           UINT32            * ResultPtr
                           )
```

4.4.8.3 Arguments

Data Type	Formal Name	Description	Valid Values
pAMD_FLASH_B_t	AMD_FLASHPtr	AMD_FLASH_B base address associated with this driver.	Any non-zero core processor data address.
UINT32 *	ResultPtr	Result address in which to write device ID	Any non-zero core processor data address.

4.4.8.4 Return Values

Error Code	Description
AMD_FLASH_B_ERR_NONE	No error
AMD_FLASH_B_ERR_BAD_RESULT_ADDR	Result address is NULL

4.4.9 AMD_FLASH_B_VerifySectorProtect

4.4.9.1 Description

AMD_FLASH_B_VerifySectorProtect specifies whether a given sector is protected or not. This function writes to the flash command addresses 2 and 1. Parameter checking is a compile-time option.

4.4.9.2 Prototype

```
AMD_FLASH_B_ReturnCode_t AMD_FLASH_B_VerifySectorProtect(
                                    pAMD_FLASH_B_tAMD_FLASHPtr,
                                    AMD_FLASH_B_SECTOR_t Sector,
                                    AMD_FLASH_B_Boolean_t * ResultPtr
                                    )
```

4.4.9.3 Arguments

Data Type	Formal Name	Description	Valid Values
pAMD_FLASH_B_t	AMD_FLASHPtr	AMD_FLASH_B base address associated with this driver.	Any non-zero core processor data address.
AMD_FLASH_B_SECTOR_t	Sector	The sector on which to check the protection.	AMD_FLASH_B_SA0 through AMD_FLASH_B_SA18
AMD_FLASH_B_Boolean_t *	ResultPtr	Result address in which to write verify information	AMD_FLASH_B_TRUE = Protection on AMD_FLASH_B_FALSE = Protection off

4.4.9.4 Return Values

Error Code	Description
AMD_FLASH_B_ERR_NONE	No error
AMD_FLASH_B_ERR_BAD_RESULT_ADDR	Result address is NULL
AMD_FLASH_B_ERR_INVALID_SECTOR	Sector given not valid.

4.4.10 AMD_FLASH_B_BlankCheck

4.4.10.1 Description

AMD_FLASH_B_BlankCheck determines whether the data in the address specified by the user is blank. Parameter checking is a compile-time option.

4.4.10.2 Prototype

```
AMD_FLASH_B_ReturnCode_t AMD_FLASH_B_BlankCheck(
                           pAMD_FLASH_B_t    AMD_FLASHPtr,
                           UINT32            * TargetPtr,
                           AMD_FLASH_B_Boolean_t * BlankPtr
)
```

4.4.10.3 Arguments

Data Type	Formal Name	Description	Valid Values
pAMD_FLASH_B_t	AMD_FLASHPtr	AMD_FLASH_B base address associated with this driver	Any non-zero core processor data address.
UINT32 *	TargetPtr	Target address in which blankcheck is to occur	Any non-zero flash address.
AMD_FLASH_B_Boolean_t *	BlankPtr	Result address in which to write blankcheck results	AMD_FLASH_B_TRUE = Blank AMD_FLASH_B_FALSE = Not blank

4.4.10.4 Return Values

Error Code	Description
AMD_FLASH_B_ERR_NONE	No error
AMD_FLASH_B_ERR_BAD_RESULT_ADDR	Result address is zero

4.4.11 AMD_FLASH_B_GetStatus

4.4.11.1 Description

AMD_FLASH_B_GetStatus determines whether the data in the address specified by the user is written as specified. Parameter checking is a compile-time option.

4.4.11.2 Prototype

```
AMD_FLASH_B_ReturnCode_t AMD_FLASH_B_GetStatus (
          pAMD_FLASH_B_t              AMD_FLASHPtr,
          UINT32                    * TargetPtr,
          UINT32                      ExpectedVal,
          AMD_FLASH_B_Boolean_t     * ResultPtr
)
```

4.4.11.3 Arguments

Data Type	Formal Name	Description	Valid Values
pAMD_FLASH_B_t	AMD_FLASHPtr	AMD_FLASH_B base address associated with this driver.	Any non-zero core processor data address.
UINT32 *	TargetPtr	Target address in which verification occurs	Any flash address.
UINT32	ExpectedVal	Expected data value in TargetPtr	Any 32-bit value.
AMD_FLASH_B_Boolean_t *	ResultPtr	Result address in which to write verification results	AMD_FLASH_B_TRUE = Data matches AMD_FLASH_B_FALSE = Data does not match

4.4.11.4 Return Values

Error Code	Description
AMD_FLASH_B_ERR_NONE	No error
AMD_FLASH_B_ERR_BAD_RESULT_ADDR	Result address is zero

4.5 AMD_FLASH_B Example Application

The following describes the AMD_FLASH_B example application.

4.5.1 Description for Example Application

The AMD_FLASH_B example program illustrates use of the AMD_FLASH_B driver. It operates on an M•CORE MMCCMB3401. The program demonstrates how to verify manufacturer and device identification, erase sectors and the entire chip, program normally and program in unlock bypass mode. The program assumes the flash is on an MMCCMB3401. The only global variables used are a pointer to the flash register block. Status of errors is checked by return codes. If an API call returns an abnormal status the program returns with the line number at which it failed as the argument and program operation terminates. The program returns on its own at completion.

4.5.1.1 Example Application Code

```
/*-------------------------------------------------------------------------------
-
Example Application for AMD_FLASH_B Module

File: amd_flashb_app.c

Purpose: This program executes an example application of the M-CORE Peripherals
Library AMD_FLASH_B Level 1 driver for the AMD Am29LV800. It operates on an MCORE
MMCCMB3401. The program demonstrates how to verify manufacturer and device
identification, erase sectors and the entire chip, program normally and program in
unlock bypass mode. The program  returns on its own at completion. If an error occurs,
the program returns the line number of the failing test.

Time to Run: Approximately 23 seconds

General Setup: General hardware and software setup to run this application can be
found in the 'readme_app.txt' file, located in the '<library>/app' directory.

Application Specific Setup: The program requires the processor to operate in
supervisor mode.

(C) Copyright Motorola Inc, 1999. All rights reserved.
-------------------------------------------------------------------------------*
/

#include <stdlib.h>
#include "amd_flash_b.h"
#include "errors.h"

/* macro to handle error checking and response */
#define handle_error(err) if (err != AMD_FLASH_B_ERR_NONE) return(__LINE__);

int main(void);                              /* function prototype */
int amdflash_example(pAMD_FLASH_B_t);        /* function prototype */
```

```
/*****************************************************************************
*
Name: main
Description: Drive program operation
Inputs: amdptr (pointer to AMD_FLASH_B base address)
Outputs: none
*****************************************************************************
/
int main ()
{
   volatile int status;

   status = amdflash_example (__MMCCMB3401_MLB_AMD_FLASH);

   return (0);
}

/*****************************************************************************
*
Name: amdflash_example
Description: Provide an example for the usage of the AMD_FLASH_B
Inputs: flashptr (pointer to AMD_FLASH_A base address)
Outputs: exit status (if passed, 0; else line number of the failure)
*****************************************************************************
/
int amdflash_example (pAMD_FLASH_B_t flashptr)
{
   AMD_FLASH_B_ReturnCode_t retval;          /* error return code */
   UINT32 base,                              /* base address */
   word_size = 4,
   count,
   count_max = 4,
   resultptr,
   *sector0,
   *sector5,
   sector0ptr[4],
   sector5ptr[4],
   programval[4] =
   {0x5A5A5A5A, 0x1E1E1E1E, 0xFF00FF00, 0x12345678};
   AMD_FLASH_B_Boolean_t boolptr;            /* status indicator */

   base = (UINT32)flashptr;
   sector0 = (UINT32 *)base + AMD_FLASH_B_SA0_BOTTOM_BOOT;
   sector5 = (UINT32 *)base + AMD_FLASH_B_SA5_BOTTOM_BOOT;

   /* initialize arrays with addresses to be programmed */
   for (count = 0; count < count_max; count++)
   {
   sector0ptr[count] = (UINT32)sector0 + (count * word_size);

   sector5ptr[count] = (UINT32)sector5 + (count * word_size);
   }

   /* get manufacturer ID */
```

AMD_FLASH_B Level 1

```
   retval = AMD_FLASH_B_GetManufacturerID(flashptr, /* base addr. */
                                    &resultptr);/* address to */
                                             /* store ID */
   handle_error(retval);

   if (resultptr != AMD_MANUFACTURER_ID_WORD)
   {
   return(__LINE__);
   }

   /* get device ID */
   retval = AMD_FLASH_B_GetDeviceID(flashptr,/* base address */
                           &resultptr);      /* address to store*/
                                             /* device ID */
   handle_error(retval);

   /* erase sector 0 */
   retval = AMD_FLASH_B_Erase(flashptr,      /* base address */
                           AMD_FLASH_B_SA0); /* sector to erase */
   handle_error(retval);

   /* wait until erase completes */
   do
   {
   retval = AMD_FLASH_B_BlankCheck(flashptr, /* base address */
                           sector0,          /* address to check*/
                           &boolptr);        /* address to store*/
                                             /* check results */
   handle_error(retval);
   } while (boolptr == AMD_FLASH_B_FALSE);

   /* program four addresses in sector 0 */
   for (count = 0; count < count_max; count++)
   {
   /* write the value */
   retval = AMD_FLASH_B_Program(flashptr,    /* base address */
                  (UINT32 *)sector0ptr[count],/* address to program */
                  programval[count]);        /* program data value */
                  handle_error(retval);      /* wait until program completes */
   do
   {
   retval = AMD_FLASH_B_GetStatus(flashptr,  /* base address */
                  (UINT32 *)sector0ptr[count],/* address to check */
                  programval[count],         /* value expected */
                  &boolptr);                 /* address to store */
                                             /* check results */
   handle_error(retval);
   } while (boolptr == AMD_FLASH_B_FALSE);
} /* end program for loop */

   /* erase sector 5 */
   retval = AMD_FLASH_B_Erase(flashptr,      /* base address */
                           AMD_FLASH_B_SA5); /* sector to erase */
   handle_error(retval);
```

```
   /* wait until erase completes */
   do
   {
   retval = AMD_FLASH_B_BlankCheck(flashptr, /* base address */
                           sector5,          /* address to check*/
                           &boolptr);        /* address to store*/
   /* check results */
   handle_error(retval);
   } while (boolptr == AMD_FLASH_B_FALSE);

   /* enable unlock bypass mode */
   retval = AMD_FLASH_B_ControlUnlockBypass(flashptr,/* base address */
                                            AMD_FLASH_B_TRUE);/* bypass mode on */
   handle_error(retval);

   /* program four addresses in sector 5 */
   for (count = 0; count < count_max; count++)
   {
   /* write the value */
   retval = AMD_FLASH_B_ProgramUnlockBypass(flashptr,/* base address */
                           (UINT32 *)sector5ptr[count],/* address to program */
                           programval[count]);/* program data value */
   handle_error(retval);

   /* wait until program completes */
   do
   {
   retval = AMD_FLASH_B_GetStatus(flashptr,  /* base address */
                           (UINT32 *)sector5ptr[count],/* address to check */
                           programval[count],/* program data value */
                           &boolptr);
   handle_error(retval);
   } while (boolptr == AMD_FLASH_B_FALSE);
} /* end program for loop */

   retval = AMD_FLASH_B_Erase(flashptr,      /* base address */
                           AMD_FLASH_B_ALL); /* erase all sectors */
   handle_error(retval);

   /* wait until erase completes */
   do
   {
   retval = AMD_FLASH_B_BlankCheck(flashptr, /* base address */
                           sector0,          /* address to check*/
                           &boolptr);        /* address to store*/
                                             /* check results */
   handle_error(retval);
   } while (boolptr == AMD_FLASH_B_FALSE);

return (0);

} /* end main */
```

AMD_FLASH_B Level 1

Section 5 CCM_A Level 1

This section describes the CCM_A Level 1 M•CORE™ Device Driver.

5.1 Overview

The Chip Configuration Module (CCM) and its accompanying driver (CCM_A) are used on the MMC2107 development system. The level 1 service is designed to allow the programmer to control the CCM on a near-register level, but without the concern of exact positions of registers and fields, or the detailed sequences of register operations. Refer to the MMC2107 Advance Information (Data Book) (MMC2107/D) for additional information.

5.2 CCM_A Service Functions

The CCM_A device driver software provides these functions:

- Initializes the chip configuration module (CCM).
- Obtains the chip mode.
- Determines how the bus monitor responds during debug mode and selects the time-out time for the bus monitor.
- Obtains the default operation of certain chip functions.
- Enables and disables the TSIZ and PSTAT signal enables.
- Obtains the chip part identification and revision numbers.
- Writes unformatted data to the Chip Configuration Register (CCR).
- Gets unformatted data from selected chip configuration registers.

5.3 CCM_A Data Type Definitions

The following paragraphs describe the data definitions recognized by the CCM_A device driver. These data definitions are grouped into six categories: standard data types, board specific base address definitions, derived data types, enumerated data types, register macros, and return codes.

5.3.1 Standard Data Type Definitions

- UINT8 — 8-bit unsigned data value
- INT8 — 8-bit signed data value
- UINT16 — 16-bit unsigned data value

- INT16 — 16-bit signed data value
- UINT32 — 32-bit unsigned data value
- INT32 — 32-bit signed data value

5.3.2 Board Specific Base Address Definitions

```
#define __MMC2107_CCM   (0x00C10000)
```

5.3.3 Derived Data Type Definitions

5.3.3.1 CCM_A_ChipIdentificationRegister_t - Chip Identification Register Definition

```
typedef struct{
          UINT16    PartIdentificationNumber;
          UINT16    PartRevisionNumber;
} CCM_A_ChipIdentificationRegister_t,  *pCCMA_ChipIdentification_t;
```

5.3.3.2 CCM_A_ResetConfiguration_t - Reset Configuration Register Get Reset Status Definition

```
typedef struct{
         CCM_A_DefaultChipConfigurationMode_t    ModeValue;
         CCM_A_BootSelect_t                      BootSelectValue;
         CCM_A_BootSizePort_t                    BootPortSizeValue;
         CCM_A_PadDriverControl_t                PadDriverLoadValue;
         CCM_A_PLLReference_t                    PLLReferenceValue;
         CCM_A_PLLModeSelect_t                   PLLModeSelectValue;
} CCM_A_ResetConfiguration_t,  *pCCMA_ResetConfiguration_t;
```

5.3.3.3 CCM_A_t - Chip Configuration Module Register Definition

```
typedef struct {
         UINT16    CCR;        /* Chip configuration register  */
         UINT16    RESERVED0;  /* Reserved                     */
         UINT16    RCON;       /* Reset configuration register */
         UINT16    CIR;        /* Chip Identification Register */
} CCM_A_t, *pCCM_A_t;
```

5.3.4 Enumerated Data Type Definitions

5.3.4.1 CCM_A_ChipConfigurationMode_t - Chip Configuration Mode Selection

```
typedef enum {
     CCM_A_MASTER_MODE,
     CCM_A_SINGLE_CHIP_MODE,
     CCM_A_EMULATION_MODE
}CCM_A_ChipConfigurationMode_t;
```

5.3.4.2 CCM_A_DefaultChipConfigurationMode_t - Default Chip Configuration Mode Selection

```
typedef enum {
     CCM_A_MASTER_MODE,
     CCM_A_SINGLE_CHIP_MODE
}CCM_A_DefaultChipConfigurationMode_t;
```

5.3.4.3 CCM_A_BusMonitorTiming_t - Bus Monitor Timing Selection

```
typedef enum {
     CCM_A_TIME_OUT_64,
     CCM_A_TIME_OUT_32,
     CCM_A_TIME_OUT_16,
     CCM_A_TIME_OUT_8
}CCM_A_BusMonitorTiming_t;
```

5.3.4.4 CCM_A_BusMonitorDebugMode_t - Bus Monitor Debug Mode Selection

```
typedef enum {
     CCM_A_DISABLE_BUS_MONITOR,   /* Disable   */
     CCM_A_ENABLE_BUS_MONITOR     /* Enable    */
} CCM_A_BusMonitorDebugMode_t;
```

5.3.4.5 CCM_A_BusMonitorExternal_t - Bus Monitor External Enable Selection

```
typedef enum {
     CCM_A_DISABLE_FOR_EXTERNAL_BUS_CYCLE,   /* Disable   */
     CCM_A_ENABLE_FOR_EXTERNAL_BUS_CYCLE     /* Enable    */
} CCM_A_BusMonitorExternal_t;
```

5.3.4.6 CCM_A_ShowInterrupt_t - Show Interrupt Selection

```
typedef enum {
     CCM_A_NORMAL_FUNCTION,               /* RSTOUT functions normally     */
     CCM_A_INTERNAL_INTERRUPT_INDICATION  /* Internal interrupt indication */
                                          /* through external RSTOUT pin   */
} CCM_A_ShowInterrupt_t;
```

5.3.4.7 CCM_A_PSTATSignal_t - PSTAT Signal Enable Selection

```
typedef enum {
     CCM_A_PSTAT_FUNCTION_DISABLED,   /* Pins configured for primary function */
     CCM_A_PSTAT_FUNCTION_ENABLED     /* Enabled                               */
} CCM_A_PSTATSignal_t;
```

5.3.4.8 CCM_A_TSIZSignal_t - TSIZ Signal Enable Selection

```
typedef enum {
     CCM_A_TSIZ_FUNCTION_DISABLED,   /* Pins configured for primary function */
     CCM_A_TSIZ_FUNCTION_ENABLED     /* Enabled                               */
} CCM_A_TSIZSignal_t;
```

5.3.4.9 CCM_A_EmulateInternalAddressSpace_t - Emulate Internal Address Space using CS1 Selection

```
typedef enum {
     CCM_A_CS1_DECODES_EXTERNAL_MEMORY,   /* External memory space decoded   */
     CCM_A_CS1_DECODES_INTERNAL_MEMORY    /*  Internal memory space decoded  */
} CCM_A_EmulateInternalAddressSpace_t;
```

5.3.4.10 CCM_A_ShowCycleEnable_t - Show Cycle Enable Selection

```
typedef enum {
     CCM_A_SHOW_CYCLES_DISABLED,        /* Enable Show Cycles     */
     CCM_A_SHOW_CYCLES_ENABLED          /*  Disable Show Cycles   */
} CCM_A_ShowCycleEnable_t;
```

5.3.4.11 CCM_A_BootSelect_t - Boot Selection

```
typedef enum {
     CCM_A_INTERNAL_BOOT_DEVICE,         /*  Internal boot  */
     CCM_A_EXTERNAL_BOOT_DEVICE          /*  External boot  */
} CCM_A_BootSelect_t;
```

5.3.4.12 CCM_A_BootSizePort_t - Boot Size Port Selection

```
typedef enum {
     CCM_A_16_BIT_PORT,            /*  16 Bit Port  */
     CCM_A_32_BIT_PORT             /*  32 Bit Port  */
} CCM_A_BootSizePort_t;
```

5.3.4.13 CCM_A_PadDriverLoad_t - Pad Driver Load Selection

```
typedef enum {
     CCM_A_DEFAULT_DRIVE_STRENGTH,       /*  Default Drive Strength  */
     CCM_A_FULL_DRIVE_STRENGTH           /*  Full Drive Strength     */
} CCM_A_PadDriverLoad_t;
```

5.3.4.14 CCM_A_PLLReference_t - PLL Reference Selection

```
typedef enum {
     CCM_A_EXTERNAL_CLOCK,           /*  External clock      */
     CCM_A_CRYSTAL_OSCILLATOR        /*  Crystal Oscillator  */
} CCM_A_PLLReference_t;
```

5.3.4.15 CCM_A_PLLModeSelect_t - PLL Mode Selection

```
typedef enum {
     CCM_A_1_TO_1_PLL_MODE,           /* 1:1 PLL mode      */
     CCM_A_NORMAL_PLL_MODE            /* Normal PLL mode   */
} CCM_A_PLLModeSelect_t;
```

5.3.4.16 CCM_A_Register_t - Chip Configuration Module Register Selection

```
typedef enum {
     CCM_A_CCR,        /* Select Chip configuration register  */
     CCM_A_RCON,       /* Select Reset configuration register */
     CCM_A_CIR         /* Select Chip Identification Register */
} CCM_A_Register_t;
```

5.3.4.17 CCM_A_ReturnCode_t - Error Return Code Selection

```
typedef enum{
     CCM_A_ERR_NONE,
     CCM_A_ERR_INVALID_HANDLE,
     CCM_A_ERR_BAD_RESULT_ADDR,
     CCM_A_ERR_INVALID_REGISTER,
     CCM_A_ERR_INVALID_LOAD_VALUE,
     CCM_A_ERR_INVALID_SHOW_CYCLE_VALUE,
     CCM_A_ERR_INVALID_EMULATE_ADDRESS_VALUE,
     CCM_A_ERR_INVALID_TSIZ_SIGNAL_VALUE,
     CCM_A_ERR_INVALID_PSTAT_SIGNAL_VALUE,
     CCM_A_ERR_INVALID_SHOW_INTERRUPT_VALUE,
     CCM_A_ERR_INVALID_BUS_MONITOR_DEBUG_MODE_VALUE,
     CCM_A_ERR_INVALID_BUS_MONITOR_VALUE,
     CCM_A_ERR_INVALID_BUS_MONITOR_TIMING_VALUE
} CCM_A_ReturnCode_t;                              /* CCM_A return codes */
```

5.3.5 Register Macros

5.3.5.1 Chip Configuration Register

```
#define CCR_LOAD_BITNO 15
#define CCR_LOAD_MASK ( 1 << CCR_LOAD_BITNO )
#define CCR_SHEN_BITNO 13
#define CCR_SHEN_MASK (1 << CCR_SHEN_BITNO)
#define CCR_EMINT_BITNO 12
#define CCR_EMINT_MASK (1 << CCR_EMINT_BITNO)
#define CCR_MODE_MAX 0x7
#define CCR_MODE_BITNO 8
#define CCR_MODE_MASK (CCR_MODE_MAX << CCR_MODE_BITNO)
#define CCR_SZEN_BITNO 6
#define CCR_SZEN_MASK (1 << CCR_SZEN_BITNO)
#define CCR_PSTEN_BITNO 5
#define CCR_PSTEN_MASK (1 << CCR_PSTEN_BITNO)
#define CCR_SHINT_BITNO 4
#define CCR_SHINT_MASK ( 1 << CCR_SHINT_BITNO)
#define CCR_BME_BITNO 3
#define CCR_BME_MASK (1 << CCR_BME_BITNO)
#define CCR_BMD_BITNO 2
#define CCR_BMD_MASK (1 << CCR_BMD_BITNO)
#define CCR_BMT_MAX 0x3
#define CCR_BMT_BITNO 0
#define CCR_BMT_MASK (CCR_BMT_MAX << CCR_BMT_BITNO)
```

```
#define CCR_SINGLE_CHIP_MODE_RESET_MASK 0x0608
#define CCR_MASTER_MODE_RESET_MASK 0x0748
#define CCR_EMULATION_MODE_RESET_MASK 0x3068

/* Chip Configuration Register (CCR) Mode Field Values     */
#define CCR_MASTER_MODE_MASK  (0x7 << CCR_MODE_BITNO)
#define CCR_SINGLE_CHIP_MODE_MASK (0x6 << CCR_MODE_BITNO)
#define CCR_FAST_MODE_MIN_MASK   (0x4 << CCR_MODE_BITNO)
#define CCR_FAST_MODE_MAX_MASK  (0x5 << CCR_MODE_BITNO)
#define CCR_EMULATION_MODE_MIN_MASK (0x0 << CCR_MODE_BITNO)
#define CCR_EMULATION_MODE_MAX_MASK (0x3 << CCR_MODE_BITNO)
```

5.3.5.2 Chip Identification Register

```
#define CIR_PIN_MAX 0xFF
#define CIR_PIN_BITNO 8
#define CIR_PIN_MASK (CIR_PIN_MAX << CIR_PIN_BITNO)
#define CIR_PRN_MAX 0xFF
#define CIR_PRN_BITNO 0
#define CIR_PRN_MASK (CIR_PRN_MAX << CIR_PRN_BITNO)
```

5.3.5.3 Reset Configuration Register

```
#define RCON_PLLSEL_BITNO 7
#define RCON_PLLSEL_MASK (1  << RCON_PLLSEL_BITNO)
#define RCON_PLLREF_BITNO 6
#define RCON_PLLREF_MASK (1  << RCON_PLLREF_BITNO)
#define RCON_LOAD_BITNO 5
#define RCON_LOAD_MASK (1  << RCON_LOAD_BITNO)
#define RCON_BOOTPS_BITNO 3
#define RCON_BOOTPS_MASK (1  << RCON_BOOTPS_BITNO)
#define RCON_BOOTSEL_BITNO 2
#define RCON_BOOTSEL_MASK (1  << RCON_BOOTSEL_BITNO)
#define RCON_MODE_BITNO 0
#define RCON_MODE_MASK (1  << RCON_MODE_BITNO)
```

5.3.6 Return Codes

Return codes from the enumerated type “CCM_A_ReturnCode_t”.

Return Code	Functions Returning	Description
CCM_A_ERR_NONE	All	No error.
CCM_A_ERR_INVALID_HANDLE	All	CCM base address parameter is zero.
CCM_A_ERR_BAD_RESULT_ADDR	GetRegister, GetChipMode, GetResetStatus, GetChipIdentification	Result Pointer is zero.
CCM_A_ERR_INVALID_REGISTER	GetRegister	CCM Register Selection switch is invalid.

Return Code	Functions Returning	Description
CCM_A_ERR_INVALID_LOAD_VALUE	CCM_A_Init	Load value is out of range.
CCM_A_ERR_INVALID_SHOW_CYCLE_VALUE	CCM_A_Init	Show cycle value is out of range.
CCM_A_ERR_INVALID_EMULATE_ADDRESS_VALUE	CCM_A_Init	Emulate Address Space value is out of range.
CCM_A_ERR_INVALID_TSIZ_SIGNAL_VALUE	CCM_A_Init, ConfigureSignalOperation	TSIZ signal enable value is out of range.
CCM_A_ERR_INVALID_PSTAT_SIGNAL_VALUE	CCM_A_Init, ConfigureSignalOperation	PSTAT signal enable value is out of range.
CCM_A_ERR_INVALID_SHOW_INTERRUPT_VALUE	CCM_A_Init	Show Interrupt value is out of range.
CCM_A_ERR_INVALID_BUS_MONITOR_DEBUG_MODE_VALUE	CCM_A_Init, ConfigureBusMonitor	Bus Monitor Debug mode value is out of range.
CCM_A_ERR_INVALID_BUS_MONITOR_VALUE	CCM_A_Init	Bus Monitor Enable value is out of range.
CCM_A_ERR_INVALID_BUS_MONITOR_TIMING_VALUE	CCM_A_Init, ConfigureBusMonitor	Bus Monitor Timing value is out of range.

5.4 CCM_A API Definitions

5.4.1 CCM_A_Init

5.4.1.1 Description

The CCM_A_Init function initializes the Chip Configuration Module (CCM). Parameter checking is a compile-time option.

5.4.1.2 Prototype

```
CCM_A_ReturnCode_t CCM_A_Init (
          pCCM_A_t                              CCMPtr,
          CCM_A_PadDriverLoad_t                 LoadVal,
          CCM_A_ShowCycleEnable_t               ShowCycleEnableVal,
          CCM_A_EmulateInternalAddressSpace_t   EmulateAddressSpaceVal,
          CCM_A_TSIZSignal_t                    TSIZSignalVal,
          CCM_A_PSTATSignal_t                   PSTATSignalVal,
          CCM_A_ShowInterrupt_t                 ShowInterruptVal,
          CCM_A_BusMonitorExternal_t            BusMonitorVal,
          CCM_A_BusMonitorDebugMode_t           BusMonitorDebugModeVal,
          CCM_A_BusMonitorTiming_t              BusMonitorTimingVal
)
```

5.4.1.3 Arguments

Data Type	Formal Name	Description	Valid Values
pCCM_A_t	CCMPtr	Chip Configuration Module base address associated with this driver.	Any non-zero core processor data address.
CCM_A_PadDriverLoad_t	LoadVal	Controls drive strength of all pad output drivers.	CCM_A_DEFAULT_DRIVE_STRENGTH CCM_A_FULL_DRIVE_STRENGTH
CCM_A_ShowCycleEnable_t	ShowCycleEnableVal	Determines whether to drive the external bus during internal transfer operations.	CCM_A_SHOW_CYCLES_DISABLED CCM_A_SHOW_CYCLES_ENABLED
CCM_A_EmulateInternalAddressSpace_t	EmulateAddressSpaceVal	Determines whether to match from the internal or external memory address space.	CCM_A_CS1_DECODES_EXTERNAL_MEMORY CCM_A_CS1_DECODES_INTERNAL_MEMORY
CCM_A_TSIZSignal_t	TSIZSignalVal	TSIZ signal enable.	CCM_A_TSIZ_FUNCTION_DISABLED CCM_A_TSIZ_FUNCTION_ENABLED
CCM_A_PSTATSignal_t	PSTATSignalVal	PSTAT signal enable.	CCM_A_PSTAT_FUNCTION_DISABLED CCM_A_PSTAT_FUNCTION_ENABLED
CCM_A_ShowInterrupt_t	ShowInterruptVal	Allows visibility of any active interrupt request.	CCM_A_NORMAL_FUNCTION CCM_A_INTERNAL_INTERRUPT_INDICATION
CCM_A_BusMonitorExternal_t	BusMonitorVal	Bus Monitor operation during external bus cycles.	CCM_A_DISABLE_FOR_EXTERNAL_BUS_CYCLE, CCM_A_ENABLE_FOR_EXTERNAL_BUS_CYCLE
CCM_A_BusMonitorDebugMode_t	BusMonitorDebugModeVal	Bus Monitor response during debug mode.	CCM_A_DISABLE_BUS_MONITOR CCM_A_ENABLE_BUS_MONITOR
CCM_A_BusMonitorTiming_t	BusMonitorTimingVal	Time-out time for the Bus Monitor.	CCM_A_TIME_OUT_64 CCM_A_TIME_OUT_32 CCM_A_TIME_OUT_16 CCM_A_TIME_OUT_8

5.4.1.4 Return Values

Error Code	Description
CCM_A_ERR_NONE	No error
CCM_A_ERR_INVALID_HANDLE	Chip Configuration Module base address parameter is zero.
CCM_A_ERR_INVALID_LOAD_VALUE	Load value is out of range.
CCM_A_ERR_INVALID_SHOW_CYCLE_VALUE	Show cycle value is out of range.
CCM_A_ERR_INVALID_EMULATE_ADDRESS_VALUE	Emulate Address Space value is out of range.
CCM_A_ERR_INVALID_TSIZ_SIGNAL_VALUE	TSIZ signal value is out of range.
CCM_A_ERR_INVALID_PSTAT_SIGNAL_VALUE	PSTAT signal value is out of range.
CCM_A_ERR_INVALID_SHOW_INTERRUPT_VALUE	Show Interrupt value is out of range.
CCM_A_ERR_INVALID_BUS_MONITOR_DEBUG_MODE_VALUE	Bus Monitor Debug mode value is out of range.

CCM_A_ERR_INVALID_BUS_MONITOR_VALUE	Bus Monitor Enable value is out of range.
CCM_A_ERR_INVALID_BUS_MONITOR_TIMING_VALUE	Bus Monitor Timing value is out of range.

5.4.2 CCM_A_GetChipMode

5.4.2.1 Description

The CCM_A_GetChipMode function obtains the chip mode.

This function should only be called after the chip configuration has been initialized through a call to the CCM_A_Init function.

5.4.2.2 Prototype

```
CCM_A_ReturnCode_t CCM_A_GetChipMode (
          pCCM_A_t                          CCMPtr,
          CCM_A_ChipConfigurationMode_t    *ChipModePtr
)
```

5.4.2.3 Arguments

Data Type	Formal Name	Description	Valid Values
pCCM_A_t	CCMPtr	CCM base address associated with this driver.	Any non-zero core processor data address.
CCM_A_ChipConfigurationMode_t	*ChipModePtr	Storage for chip mode value.	CCM_A_MASTER_MODE, CCM_A_SINGLE_CHIP_MODE, CCM_A_FAST_MODE, CCM_A_EMULATION_MODE

5.4.2.4 Return Values

Error Code	Description
CCM_A_ERR_NONE	No error
CCM_A_ERR_INVALID_HANDLE	CCM base address parameter is zero
CCM_A_ERR_BAD_RESULT_ADDR	Result Pointer is zero.

5.4.3 CCM_A_ConfigureBusMonitor

5.4.3.1 Description

The CCM_A_ConfigureBusMonitor function specifies whether the bus monitor will respond during debug mode. The function also selects the time-out time for the bus monitor.

This function should only be called after the chip configuration has been initialized through a call to the CCM_A_Init function.

5.4.3.2 Prototype

```
CCM_A_ReturnCode_t CCM_A_ConfigureBusMonitor (
          pCCM_A_t                      CCMPtr,
          CCM_A_BusMonitorDebugMode_t   BusMonitorDebugModeVal,
          CCM_A_BusMonitorTiming_t      BusMonitorTimingVal
)
```

5.4.3.3 Arguments

Data Type	Formal Name	Description	Valid Values
pCCM_A_t	CCMPtr	CCM base address associated with this driver.	Any non-zero core processor data address.
CCM_A_BusMonitorDebugMode_t	BusMonitorDebugModeVal	Bus Monitor response during debug mode.	CCM_A_DISABLE_BUS_MONITOR CCM_A_ENABLE_BUS_MONITOR
CCM_A_BusMonitorTiming_t	BusMonitorTimingVal	Time-out for the Bus Monitor.	CCM_A_TIME_OUT_64 CCM_A_TIME_OUT_32 CCM_A_TIME_OUT_16 CCM_A_TIME_OUT_8

5.4.3.4 Return Values

Error Code	Description
CCM_A_ERR_NONE	No error
CCM_A_ERR_INVALID_HANDLE	CCM base address parameter is zero
CCM_A_ERR_INVALID_BUS_MONITOR_DEBUG_MODE_VALUE	Bus Monitor Debug mode value is out of range.
CCM_A_ERR_INVALID_BUS_MONITOR_TIMING_VALUE	Bus Monitor Timing value is out of range.

5.4.4 CCM_A_GetResetStatus

5.4.4.1 Description

The CCM_A_GetResetStatus function obtains the default operation of certain chip functions.

This function should only be called after the chip configuration has been initialized through a call to the CCM_A_Init function.

5.4.4.2 Prototype

```
CCM_A_ReturnCode_t CCM_A_GetResetStatus (
          pCCM_A_t                      CCMPtr,
          pCCMA_ResetConfiguration_t   *ResetConfigurationStruct
)
```

5.4.4.3 Arguments

Data Type	Formal Name	Description	Valid Values
pCCM_A_t	CCMPtr	CCM base address associated with this driver.	Any non-zero core processor data address.
pCCMA_ResetConfiguration_t	ResetConfigurationStruct	Pointer to Reset Configuration Register.	Any non-zero address.
CCM_A_DefaultChipConfigurationMode_t	ModeValue	Default mode for PLL.	CCM_A_SINGLE_CHIP_MODE CCM_A_MASTER_CHIP_MODE
CCM_A_BootSelect_t	BootSelectValue	Default selection for Boot Device.	CCM_A_INTERNAL_BOOT_DEVICE CCM_A_EXTERNAL_BOOT_DEVICE
CCM_A_BootSizePort_t	BootPortSizeValue	Default for boot port size.	CCM_A_16_BIT_PORT CCM_A_32_BIT_PORT
CCM_A_PadDriverControl_t	PadDriverLoadValue	Configuration for pad driver strength.	CCM_A_DEFAULT_DRIVE_STRENGTH CCM_A_FULL_DRIVE_STRENGTH
CCM_A_PLLReference_t	PLLReferenceValue	Default reference for the PLL.	CCM_A_EXTERNAL_CLOCK CCM_A_CRYSTAL_OSCILLATOR
CCM_A_PLLModeSelect_t	PLLModeSelectValue	Default mode for the PLL.	CCM_A_1_TO_1_PLL_MODE CCM_A_NORMAL_PLL_MODE

5.4.4.4 Return Values

Error Code	Description
CCM_A_ERR_NONE	No error
CCM_A_INVALID_HANDLE	CCM base address parameter is zero
CCM_A_ERR_BAD_RESULT_ADDR	Result Pointer is zero.

5.4.5 CCM_A_ConfigureSignalOperation

5.4.5.1 Description

The CCM_A_ConfigureSignalOperation function enables and disables the TSIZ and PSTAT signal enables. This function should only be called after the chip configuration has been initialized through a call to the CCM_A_Init function.

5.4.5.2 Prototype

```
CCM_A_ReturnCode_t CCM_A_ConfigureSignalOperation (
          pCCM_A_t               CCMPtr,
          CCM_A_TSIZSignal_t     TSIZSignalVal,
          CCM_A_PSTATSignal_t    PSTATSignalVal
)
```

5.4.5.3 Arguments

Data Type	Formal Name	Description	Valid Values
pCCM_A_t	CCMPtr	CCM base address associated with this driver.	Any non-zero core processor data address.
CCM_A_TSIZSignal_t	TSIZSignalVal	TSIZ signal enable.	CCM_A_TSIZ_FUNCTION_DISABLED CCM_A_TSIZ_FUNCTION_ENABLED
CCM_A_PSTATSignal_t	PSTATSignalVal	PSTAT signal enable.	CCM_A_PSTAT_FUNCTION_DISABLED CCM_A_PSTAT_FUNCTION_ENABLED

5.4.5.4 Return Values

Error Code	Description
CCM_A_ERR_NONE	No error
CCM_A_ERR_INVALID_HANDLE	CCM base address parameter is zero
CCM_A_ERR_INVALID_TSIZ_SIGNAL_VALUE	TSIZ signal enable value is out of range.
CCM_A_ERR_INVALID_PSTAT_SIGNAL_VALUE	PSTAT signal enable value is out of range.

5.4.6 CCM_A_GetChipIdentification

5.4.6.1 Description

The CCM_A_GetChipIdentification function obtains the chip part identification and revision numbers. This function should only be called after the chip configuration has been initialized through a call to the CCM_A_Init function.

5.4.6.2 Prototype

```
CCM_A_ReturnCode_t CCM_A_GetChipIdentification (
          pCCM_A_t                        CCMPtr,
          pCCMA_ChipIdentification_t     *ChipIdentificationStruct
)
```

5.4.6.3 Arguments

Data Type	Formal Name	Description	Valid Values
pCCM_A_t	CCMPtr	CCM base address associated with this driver.	Any non-zero core processor data address.
pCCMA_ChipIdentification_t	ChipIdentification Struct	Pointer to Chip Identification Register.	Any non-zero address.
UINT8	ChipIdentification Struct->PartIdentificationNumber	Part Identification number.	Any non-zero core processor data address.
UINT8	ChipIdentification Struct->PartRevisionNumber	Part Revision Number.	Any non-zero core processor data address.

5.4.6.4 Return Values

Error Code	Description
CCM_A_ERR_NONE	No error
CCM_A_ERR_INVALID_HANDLE	CCM base address parameter is zero
CCM_A_ERR_BAD_RESULT_ADDR	Result Pointer is zero.

5.4.7 CCM_A_SetRegister

5.4.7.1 Description

The CCM_A_SetRegister function writes unformatted data to the Chip Configuration Register (CCR). Parameter checking is a compile-time option. This function can be called at any time.

Note that it is not possible to write to all chip configuration registers. The chip configuration registers CIR and RCON are read-only.

5.4.7.2 Prototype

```
CCM_A_ReturnCode_t CCM_A_SetRegister (
          pCCM_A_t      CCMPtr,
          UINT16        RegisterValue
)
```

5.4.7.3 Arguments

Data Type	Formal Name	Description	Valid Values
pCCM_A_t	CCMPtr	CCM_A base address associated with this driver.	Any non-zero core processor data address.
UINT16	RegisterValue	Data to copy into the CCR register.	Any 16-bit value.

5.4.7.4 Return Values

Error Code	Description
CCM_A_ERR_NONE	No error
CCM_A_ERR_INVALID_HANDLE	CCM base address parameter is zero

5.4.8 CCM_A_GetRegister

5.4.8.1 Description

The CCM_A_GetRegister function returns unformatted data from selected chip configuration registers. Parameter checking is a compile-time option.

This function can be called at any time.

5.4.8.2 Prototype

```
CCM_A_ReturnCode_t CCM_A_GetRegister (
          pCCM_A_t            CCMPtr,
          CCM_A_Register_t    CCMARegister,
          UINT16             *GetRegisterPtr
)
```

5.4.8.3 Arguments

Data Type	Formal Name	Description	Valid Values
pCCM_A_t	CCMPtr	CCM_A base address associated with this driver.	Any non-zero core processor data address.
CCM_A_Register_t	CCMARegister	Select among a sub-set of CCM_A registers.	CCM_A_CCR CCM_A_RCON CCM_A_CIR
UINT16 *	GetRegisterPtr	Result address for selected CCM_A register data.	Any non-zero core processor data address.

5.4.8.4 Return Values

Error Code	Description
CCM_A_ERR_NONE	No error
CCM_A_INVALID_HANDLE	CCM base address parameter is zero
CCM_A_BAD_RESULT_ADDR	Result Pointer is zero
CCM_A_ERR_INVALID_REGISTER	CCM Register Selection switch is invalid

5.5 CCM_A Example Application

The following describes the CCM_A example application.

5.5.1 Description for Example Application

This program initializes the chip configuration module, gets the chip mode, configures the bus monitor, gets the reset status, configures the signal operation, and gets the chip identification number.

5.5.1.1 Example Application Code

```
/*------------------------------------------------------------------------
...example...
```

Section 6 CORE_B Level 1

This section describes the CORE_B Level 1 M•CORE™ Device Driver.

6.1 Overview

The M•CORE™ processor (M210) and its accompanying driver (CORE_B) are part of the MMC2107 development system. The level 1 service is designed to allow the programmer to control the M•CORE™ processor on a near-register level, but without the concern of exact positions of registers and fields, or the detailed sequences of register operations. Refer to the MMC2107 Advance Information (Data Book) (MMC2107/D) for additional information.

6.2 CORE_B Service Functions

The CORE_B device driver software provides these functions:

- Initializes all processor vector table entries with a specified default vector value.
- Initializes a single vector table entry at the specified base address plus specified offset with a specified vector value.
- Sets the M-CORE processor status register to a user-specified state.
- Sets the vector base address register (VBR) to a user-specified value.
- Writes unformatted data to selected M-CORE control registers.
- Reads unformatted data from selected CORE_B registers.
- Sets M-CORE registers to their power-on reset state.

6.3 CORE_B Data Type Definitions

The following paragraphs describe the data definitions recognized by the CORE_B device driver. These data definitions are grouped into six categories: standard data types, derived data types, enumerated data types, register macros, miscellaneous macros, and return codes.

6.3.1 Standard Data Type Definitions

- UINT8 — 8-bit unsigned data value
- INT8 — 8-bit signed data value
- UINT16 — 16-bit unsigned data value
- INT16 — 16-bit signed data value
- UINT32 — 32-bit unsigned data value
- INT32 — 32-bit signed data value

6.3.2 Derived Data Type Definitions

```
typedef void *pVectorTable_t;      /* base address for vector table */
typedef UINT32 VectorValue_t;      /* vector table entry            */
```

6.3.3 Enumerated Data Type Definitions

6.3.3.1 CORE_B_ReturnCode_t – Error return codes codes for CORE_B

```
typedef enum
{
  CORE_B_ERR_NONE = 0x0L,                 /* No error (force long)     */
  CORE_B_ERR_BAD_RESULT_ADDR,             /* Result Address is NULL    */
  CORE_B_ERR_INVALID_REGISTER,            /* Invalid register selection*/
  CORE_B_ERR_INVALID_VECTOR_BASE,         /* Invalid vector base       */
  CORE_B_ERR_INVALID_VECTOR_VALUE,        /* Invalid vector entry      */
  CORE_B_ERR_INVALID_VECTOR_OFFSET,       /* Invalid vector offset     */
  CORE_B_ERR_INVALID_TRACE_MODE,          /* Invalid trace mode        */
  CORE_B_ERR_INVALID_EXCEPTION_CONTROL,   /* Invalid exception control */
  CORE_B_ERR_INVALID_INSTRUCTION_BOUNDARY,/* Invalid instr. boundary   */
  CORE_B_ERR_INVALID_INTERRUPT_CONTROL    /* Invalid interrupt control */
} CORE_B_ReturnCode_t; /* CORE_B return codes */
```

6.3.3.2 CORE_B_Register_t – CORE_B Register Selection

```
typedef enum {
  CORE_B_PSR,      /* Select Processor Status Register */
  CORE_B_VBR,      /* Select Vector Base Register */
  CORE_B_EPSR,     /* Select Exception Saved Processor Stat Reg */
  CORE_B_FPSR,     /* Select Interrupt Saved Processor Stat Reg */
  CORE_B_EPC,      /* Select Exception Saved Program Counter */
  CORE_B_FPC,      /* Select Interrupt Saved Program Counter */
  CORE_B_SS0,      /* Select Supervisor Storage Register 0 */
  CORE_B_SS1,      /* Select Supervisor Storage Register 1 */
  CORE_B_SS2,      /* Select Supervisor Storage Register 2 */
  CORE_B_SS3,      /* Select Supervisor Storage Register 3 */
  CORE_B_SS4,      /* Select Supervisor Storage Register 4 */
  CORE_B_GCR,      /* Select Global Control Register */
  CORE_B_GSR       /* Select Global Status Register */
} CORE_B_Register_t;
```

6.3.3.3 CORE_B_Vector_t – MCORE Vector Selection

```
typedef enum{
  CORE_B_RESET_VECTOR = 0,
  CORE_B_MISALIGNED_ACCESS_VECTOR,
  CORE_B_ACCESS_ERROR_VECTOR,
  CORE_B_DIVIDE_BY_ZERO_VECTOR,
  CORE_B_ILLEGAL_INSTRUCTION_VECTOR,
  CORE_B_PRIVILEGE_VIOLATION_VECTOR,
  CORE_B_TRACE_EXCEPTION_VECTOR,
```

```
CORE_B_BREAKPOINT_EXCEPTION_VECTOR,
CORE_B_UNRECOVERABLE_ERROR_VECTOR,
CORE_B_SOFT_RESET_VECTOR,
CORE_B_INT_AUTOVECTOR,
CORE_B_FINT_AUTOVECTOR,
CORE_B_HARDWARE_ACCELERATOR_VECTOR,
CORE_B_RESERVED_13_VECTOR,
CORE_B_RESERVED_14_VECTOR,
CORE_B_RESERVED_15_VECTOR,
CORE_B_TRAP_0_VECTOR,
CORE_B_TRAP_1_VECTOR,
CORE_B_TRAP_2_VECTOR,
CORE_B_TRAP_3_VECTOR,
CORE_B_RESERVED_20_VECTOR,
CORE_B_RESERVED_21_VECTOR,
CORE_B_RESERVED_22_VECTOR,
CORE_B_RESERVED_23_VECTOR,
CORE_B_RESERVED_24_VECTOR,
CORE_B_RESERVED_25_VECTOR,
CORE_B_RESERVED_26_VECTOR,
CORE_B_RESERVED_27_VECTOR,
CORE_B_RESERVED_28_VECTOR,
CORE_B_RESERVED_29_VECTOR,
CORE_B_RESERVED_30_VECTOR,
CORE_B_RESERVED_31_VECTOR,
CORE_B_RESERVED_32_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_33_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_34_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_35_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_36_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_37_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_38_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_39_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_40_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_41_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_42_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_43_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_44_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_45_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_46_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_47_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_48_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_49_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_50_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_51_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_52_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_53_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_54_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_55_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_56_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_57_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_58_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_59_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_60_VECTOR,    /* Reserved for Vectored Int. Cntlr */
```

CORE_B Level 1

```
CORE_B_RESERVED_61_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_62_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_63_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_64_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_65_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_66_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_67_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_68_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_69_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_70_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_71_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_72_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_73_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_74_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_75_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_76_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_77_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_78_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_79_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_80_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_81_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_82_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_83_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_84_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_85_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_86_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_87_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_88_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_89_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_90_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_91_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_92_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_93_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_94_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_95_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_96_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_97_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_98_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_99_VECTOR,    /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_100_VECTOR,   /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_101_VECTOR,   /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_102_VECTOR,   /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_103_VECTOR,   /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_104_VECTOR,   /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_105_VECTOR,   /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_106_VECTOR,   /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_107_VECTOR,   /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_108_VECTOR,   /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_109_VECTOR,   /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_110_VECTOR,   /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_111_VECTOR,   /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_112_VECTOR,   /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_113_VECTOR,   /* Reserved for Vectored Int. Cntlr */
CORE_B_RESERVED_114_VECTOR,   /* Reserved for Vectored Int. Cntlr */
```

```
  CORE_B_RESERVED_115_VECTOR,   /* Reserved for Vectored Int. Cntlr */
  CORE_B_RESERVED_116_VECTOR,   /* Reserved for Vectored Int. Cntlr */
  CORE_B_RESERVED_117_VECTOR,   /* Reserved for Vectored Int. Cntlr */
  CORE_B_RESERVED_118_VECTOR,   /* Reserved for Vectored Int. Cntlr */
  CORE_B_RESERVED_119_VECTOR,   /* Reserved for Vectored Int. Cntlr */
  CORE_B_RESERVED_120_VECTOR,   /* Reserved for Vectored Int. Cntlr */
  CORE_B_RESERVED_121_VECTOR,   /* Reserved for Vectored Int. Cntlr */
  CORE_B_RESERVED_122_VECTOR,   /* Reserved for Vectored Int. Cntlr */
  CORE_B_RESERVED_123_VECTOR,   /* Reserved for Vectored Int. Cntlr */
  CORE_B_RESERVED_124_VECTOR,   /* Reserved for Vectored Int. Cntlr */
  CORE_B_RESERVED_125_VECTOR,   /* Reserved for Vectored Int. Cntlr */
  CORE_B_RESERVED_126_VECTOR,   /* Reserved for Vectored Int. Cntlr */
  CORE_B_RESERVED_127_VECTOR    /* Reserved for Vectored Int. Cntlr */
} CORE_B_Vector_t;
```

6.3.3.4 CORE_B_Trace_t – MCORE Trace Mode

```
typedef enum{
  CORE_B_TRACE_NORMAL,
  CORE_B_TRACE_INSTRUCTION,
  CORE_B_TRACE_CHANGE_OF_FLOW
} CORE_B_Trace_t;
```

6.3.3.5 CORE_B_ExceptionControl_t – Exception Control

```
typedef enum{
  CORE_B_EXCEPTION_DISABLE,
  CORE_B_EXCEPTION_ENABLE
} CORE_B_ExceptionControl_t;
```

6.3.3.6 CORE_B_InterruptInstrBoundary_t – Interrupt on instruction boundary

```
typedef enum{
  CORE_B_INTERRUPT_ON_INSTRUCTION_BOUNDARIES,
  CORE_B_INTERRUPT_WITHIN_INSTRUCTION
} CORE_B_InterruptInstrBoundary_t;
```

6.3.3.7 CORE_B_InterruptControl_t – Interrupt Control

```
typedef enum{
  CORE_B_INTERRUPT_DISABLE,
  CORE_B_INTERRUPT_ENABLE_ONLY_NORMAL,
  CORE_B_INTERRUPT_ENABLE_FAST_AND_NORMAL
} CORE_B_InterruptControl_t;
```

6.3.4 Register Macros

6.3.4.1 MCORE Processor Status Register Macros

```
#define PSR_C_BITNO        0      /* Condition code/carry bit */
#define PSR_C_MASK        (1 << PSR_C_BITNO)
```

CORE_B Level 1

```
#define PSR_AF_BITNO       1      /* Alternate File Enable */
#define PSR_AF_MASK       (1 << PSR_AF_BITNO)
#define PSR_FE_BITNO       4      /* Fast Int. Enable */
#define PSR_FE_MASK       (1 << PSR_FE_BITNO)
#define PSR_IE_BITNO       6      /* Int. Enable */
#define PSR_IE_MASK       (1 << PSR_IE_BITNO)
#define PSR_IC_BITNO       7      /* Int. Control */
#define PSR_IC_MASK       (1 << PSR_IC_BITNO)
#define PSR_EE_BITNO       8      /* Exception Enable */
#define PSR_EE_MASK       (1 << PSR_EE_BITNO)
#define PSR_MM_BITNO       9      /* Misalignment Exception Mask */
#define PSR_MM_MASK       (1 << PSR_MM_BITNO)
#define PSR_SC_BITNO       10     /* Spare Control */
#define PSR_SC_MASK       (1 << PSR_SC_BITNO)
#define PSR_TC_BITNO       12     /* Translation Control */
#define PSR_TC_MASK       (1 << PSR_TC_BITNO)
#define PSR_TP_BITNO       13     /* Trace Pending */
#define PSR_TP_MASK       (1 << PSR_TP_BITNO)
#define PSR_TM_BITNO       14     /* Trace Mode */
#define PSR_TM_MAX         3
#define PSR_TM_MASK       (PSR_TM_MAX << PSR_TM_BITNO)
#define PSR_VEC_BITNO      16     /* Vector Number */
#define PSR_VEC_MAX        127
#define PSR_VEC_MASK      ( PSR_VEC_MAX << PSR_VEC_BITNO )
#define PSR_U0_BITNO       24     /* Harware Accelerator Control 0 */
#define PSR_U0_MASK       (1 << PSR_U0_BITNO)
#define PSR_U1_BITNO       25     /* Harware Accelerator Control 1 */
#define PSR_U1_MASK       (1 << PSR_U1_BITNO)
#define PSR_U2_BITNO       26     /* Harware Accelerator Control 2 */
#define PSR_U2_MASK       (1 << PSR_U2_BITNO)
#define PSR_U3_BITNO       27     /* Harware Accelerator Control 3 */
#define PSR_U3_MASK       (1 << PSR_U3_BITNO)
#define PSR_SP_BITNO       28     /* Spare bits */
#define PSR_SP_MAX         3      /* SP maximum field value */
#define PSR_SP_MASK       ( PSR_SP_MAX << PSR_SP_BITNO )
#define PSR_S_BITNO        31     /* Supervisor Mode */
#define PSR_S_MASK        (1 << PSR_S_BITNO)

#define PSR_RESET_MASK      0x8000
```

6.3.4.2 MCORE Vector Base Register Macros

```
#define VBR_VECTORBASE_BITNO    10
#define VBR_VECTORBASE_MAX      0x003FFFFF
#define VBR_VECTORBASE_MASK     (VBR_VECTORBASE_MAX << VBR_VECTORBASE_BITNO)

#define VBR_RESET_MASK      0x0000
```

6.3.5 Miscellaneous Macros

```
#define CORE_A_MAXIMUM_VECTORS 127  /*Number of Vectors in Vect Table */
```

```
#define VECTOR_ADDR_BOUNDARY 1024   /*MCORE Vector Base Addr Boundary */
```

6.3.6 Return Codes

Return codes from the enumerated type “CORE_B_ReturnCode_t”.

Return Code	Functions Returning	Description
CORE_B_ERR_NONE	All	No error.
CORE_B_ERR_BAD_RESULT_ADDR	GetRegister	Result address is zero
CORE_B_ERR_INVALID_REGISTER	GetRegister, SetRegister	Register selection is invalid.
CORE_B_ERR_INVALID_VECTOR_BASE	InitVectorTable, InitVector, SetVectorBase	Vector base is zero or not a 1KB offset.
CORE_B_ERR_INVALID_VECTOR_VALUE	InitVectorTable, InitVectorEntry	Vector value (ISR address) is zero.
CORE_B_ERR_INVALID_VECTOR_OFFSET	InitVectorEntry	Vector offset is invalid.
CORE_B_ERR_INVALID_TRACE_MODE	SetProcessorStatus	Trace mode is invalid.
CORE_B_ERR_INVALID_EXCEPTION_CONTROL	SetProcessorStatus	Exception control value is invalid.
CORE_B_ERR_INVALID_INSTRUCTION_BOUNDARY	SetProcessorStatus	Interrupt instruction boundary mode is invalid.
CORE_B_ERR_INVALID_INTERRUPT_CONTROL	SetProcessorStatus	Interrupt control value is invalid

6.4 CORE_B API Definitions

6.4.1 CORE_B_InitVectorTable

6.4.1.1 Description

The CORE_B_InitVectorTable function initializes all processor vector table entries with a specified default vector value. Parameter checking is a compile-time option.

6.4.1.2 Prototype

```
CORE_B_ReturnCode_t CORE_B_InitVectorTable (
                pVectorTable_t   vectorBasePtr,
                VectorValue_t    vectorValue
)
```

6.4.1.3 Arguments

Data Type	Formal Name	Description	Valid Values
pVectorTable_t	vectorBasePtr	Vector table base address.	Any non-zero core processor data address that is on a 1KB boundary.
VectorValue_t	vectorValue	Address of an exception or interrupt processing function	Any non-zero core processor data address

6.4.1.4 Return Values

Error Code	Description
CORE_B_ERR_NONE	No error
CORE_B_ERR_INVALID_VECTOR_BASE	Vector base is zero or not a 1KB offset.
CORE_B_ERR_INVALID_VECTOR_VALUE	Vector value (ISR address) is zero.

6.4.2 CORE_B_InitVectorEntry

6.4.2.1 Description

The CORE_B_InitVectorEntry function initializes a single vector table entry at the specified base address plus specified offset with a specified vector value. Parameter checking is a compile-time option.

6.4.2.2 Prototype

```
CORE_B_ReturnCode_t CORE_B_InitVectorEntry (
         pVectorTable_t     vectorBasePtr,
         CORE_B_Vector_t    vectorOffset,
          VectorValue_t     vectorValue
)
```

6.4.2.3 Arguments

Data Type	Formal Name	Description	Valid Values
pVectorTable_t	vectorBasePtr	Vector table base address.	Any non-zero core processor data address that is on a 1KB boundary.
CORE_B_Vector_t	vectorOffset	Vector table offset	Any enumeration of CORE_B_Vector_t
VectorValue_t	vectorValue	Address of an exception or interrupt processing function	Any non-zero core processor data address

6.4.2.4 Return Values

Error Code	Description
CORE_B_ERR_NONE	No error
CORE_B_ERR_INVALID_VECTOR_BASE	Vector base is zero or not a 1KB offset.
CORE_B_ERR_INVALID_VECTOR_VALUE	Vector value (ISR address) is zero.
CORE_B_ERR_INVALID_VECTOR_OFFSET	Vector offset is invalid.

6.4.3 CORE_B_SetProcessorStatus

6.4.3.1 Description

CORE_B_SetProcessorStatus sets the M-CORE processor status register to a user-specified state. This function can be used to enable or disable interrupts and exceptions, set trace modes, and set the interrupt instruction boundary mode. Parameter checking is a compile-time option.

6.4.3.2 Prototype

```
CORE_B_ReturnCode_t CORE_B_SetProcessorStatus (
          CORE_B_Trace_t                    traceMode,
          CORE_B_ExceptionControl_t         exceptionControl,
          CORE_B_InterruptInstrBoundary_t   instrBoundaryMode,
          CORE_B_InterruptControl_t         interruptControl
)
```

CORE_B Level 1

6.4.3.3 Arguments

Data Type	Formal Name	Description	Valid Values
CORE_B_Trace_t	traceMode	Processor trace mode	CORE_B_TRACE_NORMAL, CORE_B_TRACE_INSTRUCTION, CORE_B_TRACE_CHANGE_OF_FLOW
CORE_B_ExceptionControl_t	exceptionControl	Control selection to enable or disable exceptions	CORE_B_EXCEPTION_DISABLE, CORE_B_EXCEPTION_ENABLE
CORE_B_InterruptInstrBoundary_t	instrBoundaryMode	Interrupt instruction boundary mode	CORE_B_INTERRUPT_ON_INSTRUCTION_BOUNDARIES, CORE_B_INTERRUPT_WITHIN_INSTRUCTION
CORE_B_InterruptControl_t	interruptControl	Control selection to enable or disable interrupts	CORE_B_INTERRUPT_DISABLE, CORE_B_INTERRUPT_ENABLE_ONLY_NORMAL, CORE_B_INTERRUPT_ENABLE_FAST_AND_NORMAL

6.4.3.4 Return Values

Error Code	Description
CORE_B_ERR_NONE	No error
CORE_B_ERR_INVALID_TRACE_MODE	Trace mode is invalid.
CORE_B_ERR_INVALID_EXCEPTION_CONTROL	Exception control value is invalid.
CORE_B_ERR_INVALID_INSTRUCTION_BOUNDARY	Interrupt instruction boundary mode is invalid.
CORE_B_ERR_INVALID_INTERRUPT_CONTROL	Interrupt control value is invalid

6.4.4 CORE_B_SetVectorBase

6.4.4.1 Description

CORE_B_SetVectorBase sets the vector base address register (VBR) with a user-specified value. Parameter checking is a compile-time option.

6.4.4.2 Prototype

```
CORE_B_ReturnCode_t CORE_B_SetVectorBase (
          pVectorTable_t     vectorBasePtr
)
```

6.4.4.3 Arguments

Data Type	Formal Name	Description	Valid Values
pVectorTable_t	vectorBasePtr	Vector table base address.	Any non-zero core processor data address that is on a 1KB boundary.

6.4.4.4 Return Values

Error Code	Description
CORE_B_ERR_NONE	No error
CORE_B_ERR_INVALID_VECTOR_BASE	Vector base is not a 1KB offset.

6.4.5 CORE_B_SetRegister

6.4.5.1 Description

CORE_B_SetRegister writes unformatted data to selected M-CORE control registers. Parameter checking is a compile-time option.

6.4.5.2 Prototype

```
CORE_B_ReturnCode_t CORE_B_SetRegister (
          CORE_B_Register_t     registerSwitch,
          UINT32                registerValue
)
```

6.4.5.3 Arguments

Data Type	Formal Name	Description	Valid Values
CORE_B_Register_t	registerSwitch	Specifies a CORE_B register.	CORE_B_PSR, CORE_B_VBR, CORE_B_EPSR, CORE_B_FPSR, CORE_B_EPC, CORE_B_FPC, CORE_B_SS0, CORE_B_SS1, CORE_B_SS2, CORE_B_SS3, CORE_B_SS4, CORE_B_GCR
UINT32	registerValue	Data to copy to selected CORE_B register	Any 32-bit value.

6.4.5.4 Return Values

Error Code	Description

CORE_B_ERR_NONE	No error
CORE_B_ERR_INVALID_REGISTER	CORE_B Register Selection switch is invalid

6.4.6 CORE_B_GetRegister

6.4.6.1 Description

CORE_B_GetRegister returns unformatted data from selected CORE_B registers. Parameter checking is a compile-time option.

6.4.6.2 Prototype

```
CORE_B_ReturnCode_t CORE_B_GetRegister (
          CORE_B_Register_t     registerSwitch,
          UINT32 *              registerValuePtr
)
```

6.4.6.3 Arguments

Data Type	Formal Name	Description	Valid Values
CORE_B_Register_t	registerSwitch	Select among CORE_B registers	CORE_B_PSR, CORE_B_VBR, CORE_B_EPSR, CORE_B_FPSR, CORE_B_EPC, CORE_B_FPC, CORE_B_SS0, CORE_B_SS1, CORE_B_SS2, CORE_B_SS3, CORE_B_SS4, CORE_B_GCR, CORE_B_GSR
UINT32 *	registerValuePtr	Result address for selected CORE_B register	Any non-zero core processor data address

6.4.6.4 Return Values

Error Code	Description
CORE_B_ERR_NONE	No error
CORE_B_ERR_BAD_RESULT_ADDR	Result Pointer is zero
CORE_B_ERR_INVALID_REGISTER	CORE_B Register Selection switch is invalid

6.4.7 CORE_B_Reset

6.4.7.1 Description

CORE_B_Reset sets M-CORE registers to their power-on reset state.

6.4.7.2 Prototype

```
CORE_B_ReturnCode_t CORE_B_Reset (
)
```

6.4.7.3 Arguments

No arguments.

6.4.7.4 Return Values

Error Code	Description
CORE_B_ERR_NONE	No error

6.5 CORE_B Example Application

The following describes the CORE_B example application.

6.5.1 Description for Example Application

This program demonstrates the use of the CORE_B Level 1 Device Driver. It operates on any development board with an M210 CPU (including MMC2107, MMC2103, etc). The program initializes the M-CORE, and demonstrates how to initialize a vector table, install an interrupt handler, enable exceptions and interrupts, and process an exception. The program will return on its own at completion.

6.5.1.1 Example Application Code

```
/*------------------------------------------------------------------------
                Example Application for CORE_B Module

File: coreb_app.c

Purpose:
         This program executes an example application of the M-CORE
         Peripherals Library CORE_B Level 1 driver. It uses CORE_B API
         calls to demonstrate reading and writing to M-CORE Control
         Registers and simple exception handling.

         This program will set the vector base address, initalize
         all entries of the vector table to a default handler,
         initalize the divide by zero vector to point to a divide
         by zero handler, initalize an exception counter to zero,
         enable exceptions, force a divide by zero exception, and
         wait for the exception handle to execute.

         If an error occurs, the program will return the line number of
         where the failure occurred.

Time to Run:
         Less than one second.

General Setup:
         General hardware and software setup to run this application
         can be found in the 'readme_app.txt' file, located in the
         '<library>/app' directory.

Application Specific Setup:
         Development Board: Any development board with an M-CORE M210
         CPU (MMC2017, MMC2013, etc).

         Application requires processor to operate in supervisor mode
         with interrupts disabled and exceptions enabled.

(C) Copyright Motorola Inc, 2000.  All rights reserved.
```

```
------------------------------------------------------------------------*/

#include "core_b.h"

/************************************************************************/
/* Macro Definitions                                                    */
/************************************************************************/

#define HANDLE_ERROR(err) if (err != CORE_B_ERR_NONE) return(__LINE__)

/* example vector base address; value used here is for MMC2013 */
#define CORE_B_EXAMPLE_APP_VECTOR_BASE_ADDR 0x80804400

#undef CORE_B_PARAM_CHECKING       /* do in case of external definition */
#define CORE_B_PARAM_CHECKING 1    /* 1= enable for debug, 0 = disable  */

/************************************************************************/
/* Function Prototypes                                                  */
/************************************************************************/
static void DefaultHandler(void);            /* handler for all vectors */
static void DivideByZeroHandler(void);   /* handler for DIVBYZERO vector*/
static void DefaultHandlerAssy(void);
static void DivideByZeroHandlerAssy(void);
int coreb_example(void);
int main (void);

/************************************************************************/
/* Global Variables                                                     */
/************************************************************************/
UINT8 globalExceptionCounter = 0;

/************************************************************************/
/*  Function: main                                                      */
/*                                                                      */
/*  Purpose: Drive program operation                                    */
/*                                                                      */
/*  Inputs: none                                                        */
/*                                                                      */
/*  Outputs: status (if passed, 0; else line number of the failure)     */
/************************************************************************/
int main ( void )
{
    int status;

    status = coreb_example ( );

    return (status);
}
```

CORE_B Level 1

```
/*************************************************************************/
/* Function: coreb_example                                               */
/*                                                                       */
/* Purpose: Provide an example for the usage of the CORE_B               */
/*                                                                       */
/* Inputs:  none                                                         */
/*                                                                       */
/* Outputs: status (if passed, 0; else line number of the failure)       */
/*************************************************************************/
int coreb_example ( void )
{
/*-----------------------------------------------------------------------*/
/* Local variable declarations                                           */
/*-----------------------------------------------------------------------*/
   UINT32 RegisterContents;

   CORE_B_ReturnCode_t status;         /* device driver API return code */

   pVectorTable_t pVTable;             /* points to vector base address */
   VectorValue_t HValue;                           /* handler address */

   int a;                                            /*   temp var    */
   int b;                                            /*   temp var    */
   volatile int c;                                   /*   temp var    */

   pVTable = (void *)CORE_B_EXAMPLE_APP_VECTOR_BASE_ADDR; /* vector base*/

   RegisterContents = 0;

/*-----------------------------------------------------------------------*/
/* Set Vector Base address                                               */
/*-----------------------------------------------------------------------*/

   /* Call CORE_B_SetVectorBase with valid parameters */
   status = CORE_B_SetVectorBase( pVTable );

   HANDLE_ERROR(status);

/*-----------------------------------------------------------------------*/
/* Initalize Vector Table to default handler                             */
/*-----------------------------------------------------------------------*/
/* Call CORE_A_InitVectorTable with valid parameters */

   HValue = (VectorValue_t)&DefaultHandler;

   /* Call CORE_B_InitVectorTable with valid parameters */
   CORE_B_InitVectorTable( pVTable,
                           HValue );
   HANDLE_ERROR(status);

/*-----------------------------------------------------------------------*/
/* Initalize Divide by Zero Vector                                       */
/*-----------------------------------------------------------------------*/
```

```
   HValue = (VectorValue_t)&DivideByZeroHandler;

   /* Call CORE_B_InitVectorEntry with valid parameters */
   status = CORE_B_InitVectorEntry( pVTable,
                                    CORE_B_DIVIDE_BY_ZERO_VECTOR,
                                    HValue );
   HANDLE_ERROR(status);


/*-------------------------------------------------------------------------*/
/* Enable processor exceptions                                             */
/*-------------------------------------------------------------------------*/

   /* Call CORE_A_SetProcessorStatus with valid parameters */
   status = CORE_B_SetProcessorStatus (
                  CORE_B_TRACE_NORMAL,
               CORE_B_EXCEPTION_ENABLE,        /* enable exceptions */
               CORE_B_INTERRUPT_ON_INSTRUCTION_BOUNDARIES,
               CORE_B_INTERRUPT_DISABLE );
   HANDLE_ERROR(status);

/*-------------------------------------------------------------------------*/
/* Divide 1 by 0                                                           */
/*-------------------------------------------------------------------------*/
   a = 1;
   b = 0;
   c = a/b; /* cause divide by zero exception */

/*-------------------------------------------------------------------------*/
/* Loop until GlobalExceptionCounter is equal to one                       */
/*-------------------------------------------------------------------------*/
   while(globalExceptionCounter == 0)
   {
      /* wait for exception handler to increment GlobalExceptionCounter */
   }

/*-------------------------------------------------------------------------*/
/* Return CORE_B_ERR_NONE if no execution errors occur                     */
/*-------------------------------------------------------------------------*/
   return(0);
}

/*************************************************************************/
/*  Function: DivideByZeroHandler                                        */
/*                                                                       */
/*  Purpose: Handler for divide by zero exception                        */
/*                                                                       */
/*  Inputs: none                                                         */
/*                                                                       */
/*  Outputs: none                                                        */
/*************************************************************************/
static void DivideByZeroHandler (void)
{
    globalExceptionCounter ++;  /* increment exception counter */
```

CORE_B Level 1

```
    DivideByZeroHandlerAssy();
}

/*********************************************************************/
/*  Function: DefaultHandler                                         */
/*                                                                   */
/*  Purpose: Handler for default exception                           */
/*                                                                   */
/*  Inputs: none                                                     */
/*                                                                   */
/*  Outputs: none                                                    */
/*********************************************************************/
static void DefaultHandler (void)
{
    globalExceptionCounter ++;  /* increment exception counter */
    DefaultHandlerAssy();
}
      .export     DefaultHandlerAssy
      .export     DivideByZeroHandlerAssy

      .text

/* internal functions */
/*********************************************************************/
/* Function: DefaultHandler (assy used can compile with CodeWarrior)    */
/*                                                                   */
/* Purpose: Vector here on unexpected exceptions while running main     */
/*                                                                   */
/* Inputs & Outputs:  none                                           */
/*********************************************************************/
DefaultHandlerAssy:
  subi r0,32                /* advance stack pointer */
  stm r1-r15,(r0)           /* save register context */
  bkpt                      /* breakpoint for debugging */
  ldm r1-r15,(r0)           /* restore register context */
  addi r0,32                /* restore stack pointer */
  rte                       /* return from exception */

/*********************************************************************/
/* Function:  DivideByZeroHandler (assy used can compile with           */
/*                                 CodeWarrior)                      */
/*                                                                   */
/* Purpose: Vector here on divide by zero exception while running main  */
/*                                                                   */
/* Inputs & Outputs:  none                                           */
/*********************************************************************/
DivideByZeroHandlerAssy:

  subi r0,32                /* advance stack pointer */
  stm r1-r15,(r0)           /* save register context */
  bkpt                      /* breakpoint for debugging */
  mfcr r2,epc               /* get the exception program counter */
  addi r2,2                 /* advance to instruction after divide */
  mtcr r2,epc               /* to continue prog after rte */
```

```
ldm r1-r15,(r0)          /* restore register context */
addi r0,32               /* restore stack pointer */
rte                      /* return from exception */
```

CORE_B Level 1

Section 7 CS_A Level 1

This section describes the CS_A Level 1 M•CORE™ Device Driver.

7.1 Overview

The Chip Select module (CS) and its accompanying driver (CS_A) are a part of the MMC2107 development system. The level 1 service is designed to allow the programmer to control the CS module on a near-register level, but without the concern of exact positions of registers and fields, or the detailed sequences of register operations. Refer to the MMC2107 Advance Information (Data Book) (MMC2107/D) for additional information.

7.2 CS_A Service Functions

The CS_A device driver software provides these functions:

- Resets the chip select module to the default hardware values.
- Writes a chip select register with the user defined values requested to perform a chip select.
- Enables or disables the chip select features.
- Writes unformatted data to selected chip select module registers.
- Gets unformatted data from selected chip select module registers.

7.3 CS_A Data Type Definitions

The following paragraphs describe the data definitions recognized by the CS_A device driver. These data definitions are grouped into six categories: standard data types, board specific base address definitions, derived data types, enumerated data types, register macros, and return codes.

7.3.1 Standard Data Type Definitions

- UINT8 — 8-bit unsigned data value
- INT8 — 8-bit signed data value
- UINT16 — 16-bit unsigned data value
- INT16 — 16-bit signed data value
- UINT32 — 32-bit unsigned data value
- INT32 — 32-bit signed data value

7.3.2 Board Specific Base Address Definitions

```
#define __MMC2107_CS   (0x00C2 0000)
```

7.3.3 Derived Data Type Definitions

7.3.3.1 CS_A_t - Chip Select Module Register Definition

```
typedef struct {
           UINT16   CSCR0;     /* Chip Select Register 0 */
           UINT16   CSCR1;     /* Chip Select Register 1 */
           UINT16   CSCR2;     /* Chip Select Register 2 */
           UINT16   CSCR3;     /* Chip Select Register 3 */
} CS_A_t, *pCS_A_t;
```

7.3.4 Enumerated Data Type Definitions

7.3.4.1 CS_A_RegisterSwitch_t - Chip Select Module Register Selection

```
typedef enum {
           CS_A_CSCR0,      /* Select chip select register 0     */
           CS_A_CSCR1,      /* Select chip select register 1     */
           CS_A_CSCR2,      /* Select chip select register 2     */
           CS_A_CSCR3,      /* Select chip select register 3     */
} CS_A_RegisterSwitch_t;
```

7.3.4.2 CS_A_ReturnCode_t - Error Return Code Selection

```
typedef enum{
      CS_A_ERR_NONE,
      CS_A_ERR_INVALID_HANDLE,
      CS_A_ERR_BAD_RESULT_ADDR,
      CS_A_ERR_INVALID_REGISTER,
      CS_A_ERR_INVALID_CONTROL,
      CS_A_ERR_INVALID_TRANSFER_ACK_VAL,
      CS_A_ERR_INVALID_WRITE_ENABLE_VAL,
      CS_A_ERR_INVALID_WAIT_STATE_VAL,
      CS_A_ERR_INVALID_WRITE_WAIT_STATE_VAL,
      CS_A_ERR_INVALID_PORT_SIZE_VAL,
      CS_A_ERR_INVALID_READ_ONLY_VAL,
      CS_A_ERR_INVALID_SUPERVISOR_ONLY_VAL,
      CS_A_ERR_INVALID_CHIP_SELECT_ENABLE_VAL,
      CS_A_ERR_INVALID_RESET_CONDITION_VAL
} CS_A_ReturnCode_t; /* CS_A return codes */
```

7.3.4.3 CS_A_ChipSelectControl_t - Chip Select Enable Selection

```
typedef enum {
      CS_A_CHIP_SELECT_DISABLE,          /* Enable Chip Select    */
      CS_A_CHIP_SELECT_ENABLE            /* Disable Chip Select   */
} CS_A_ChipSelectControl_t;
```

7.3.4.4 CS_A_TransferAckControl_t - Transfer Acknowledge Selection

```
typedef enum {
      CS_A_EXTERNAL_LOGIC_ASSERTION,       /* Enable Transfer Acknowledge      */
      CS_A_CHIP_SELECT_LOGIC_ASSERTION     /* Disable Transfer Acknowledge     */
} CS_A_TransferAckControl_t;
```

7.3.4.5 CS_A_WaitStates_t - Wait State Selection

```
typedef enum {
      CS_A_WAIT_STATE_ZERO,
      CS_A_WAIT_STATE_ONE,
      CS_A_WAIT_STATE_TWO,
      CS_A_WAIT_STATE_THREE,
      CS_A_WAIT_STATE_FOUR,
      CS_A_WAIT_STATE_FIVE,
      CS_A_WAIT_STATE_SIX
} CS_A_WaitStates_t;
```

7.3.4.6 CS_A_WriteEnable_t - Write Enable Selection

```
typedef enum {
      CS_A_WRITE_AS_BYTES_ENABLE,          /* Enable Write as Bytes     */
      CS_A_EXTERNAL_WRITE_ENABLE           /* Enable Write Enable       */
} CS_A_WriteEnable_t;
```

7.3.4.7 CS_A_WriteWaitState_t - Write Wait State Selection

```
typedef enum {
      CS_A_NO_WAIT_STATE_ADD,       /* No wait state is added to write cycle    */
      CS_A_ADD_ONE_WAIT_STATE       /*  One wait state added for write cycle    */
} CS_A_WriteWaitState_t;
```

7.3.4.8 CS_A_PortSize_t - Port Size Selection

```
typedef enum {
      CS_A_PORT_SIZE_16,        /* Port Size 16   */
      CS_A_PORT_SIZE_32         /* Port Size 32   */
} CS_A_PortSize_t;
```

7.3.4.9 CS_A_ReadOnly_t - Read Only Selection

```
typedef enum {
      CS_A_READ_WRITE_ENABLE,       /* Read and write accesses are allowed   */
      CS_A_READ_ONLY_ENABLE         /* Read Only accesses allowed            */
} CS_A_ReadOnly_t;
```

7.3.4.10 CS_A_ResetCondition_t - Reset Condition Selection

```
typedef enum {
      CS_A_EMULATION_MODE,              /* Read and write accesses are allowed   */
      CS_A_16_BIT_PORT_SIZE_ENABLE,     /* Read Only accesses allowed            */
      CS_A_32_BIT_PORT_SIZE_ENABLE      /* Read Only accesses allowed            */
} CS_A_ResetCondition_t;
```

7.3.4.11 CS_A_SupervisorOnly_t - Supervisor Only Selection

```
typedef enum {
      CS_A_SUPERVISOR_AND_USER_ENABLE,       /* Supervisor and user allowed   */
      CS_A_SUPERVISOR_ONLY_ENABLE            /* Supervisor Only allowed       */
} CS_A_SupervisorOnly_t;
```

7.3.5 Register Macros

7.3.5.1 Control Register X

```
#define CSCRX_SO_BITNO 15
#define CSCRX_SO_MASK (1 << CSCRX_SO_BITNO)
#define CSCRX_RO_BITNO 14
#define CSCRX_RO_MASK (1 << CSCRX_RO _BITNO)
#define CSCRX_PS_BITNO 13
#define CSCRX_PS_MASK (1 << CSCRX_PS_BITNO)
#define CSCRX_WWS_BITNO 12
#define CSCRX_WWS_MASK (1 << CSCRX_WWS_BITNO)
#define CSCRX_WE_BITNO 11
#define CSCRX_WE_MASK (1 << CSCRX_WE_BITNO)
#define CSCRX_WS _MAX 0x7
#define CSCRX_WS_BITNO 8
#define CSCRX_WS_MASK (CSCRX_WS _MAX << CSCRX_WS_BITNO)
#define CSCRX_TAEN_BITNO 1
#define CSCRX_TAEN_MASK (CSCRX_TAEN_MAX << CSCRX_TAEN_BITNO)
#define CSCRX_CSEN_BITNO 0
#define CSCRX_CSEN_MASK (CSCRX_CSEN_MAX << CSCRX_CSEN_BITNO)
```

7.3.5.2 Control Register 0

```
#define CSCR0_RESET_MASK_EMULATON_CONDITION 0x3F02
#define CSCR0_RESET_MASK_16_BIT_PORT_CONDITION 0x1F03
#define CSCR0_RESET_MASK_32_BIT_PORT_CONDITION 0x3F03
```

7.3.5.3 Control Register 1

```
#define CSCR1_RESET_MASK_EMULATON_CONDITION 0x3F03
#define CSCR1_RESET_MASK_16_BIT_PORT_CONDITION 0x1F02
#define CSCR1_RESET_MASK_32_BIT_PORT_CONDITION 0x3F02
```

7.3.5.4 Control Register 2

```
#define CSCR2_RESET_MASK 0x3F02
```

7.3.5.5 Control Register 3

```
#define CSCR3_RESET_MASK 0x3F02
```

7.3.6 Return Codes

Return codes from the enumerated type “CS_A_ReturnCode_t”.

Return Code	Functions Returning	Description
CS_A_ERR_NONE	All	No error.
CS_A_ERR_INVALID_HANDLE	All	CS base address parameter is zero.
CS_A_ERR_BAD_RESULT_ADDR	GetRegister, GetStatus	Result Pointer is zero.
CS_A_ERR_INVALID_REGISTER	SetRegister, GetRegister	CS Register Selection switch is invalid.
CS_A_ERR_INVALID_CONTROL	ControlInterrupt	Control choice is not valid.
CS_A_ERR_INVALID_TRANSFER_ACK_VAL	CS_A_ConfigureChipSelectFeatures	Not a valid transfer acknowledge selection
CS_A_ERR_INVALID_WRITE_ENABLE_VAL	CS_A_ConfigureChipSelectFeatures	Not a valid write enable selection
CS_A_ERR_INVALID_WAIT_STATE_VAL	CS_A_ConfigureChipSelectFeatures	Not a valid wait state selection
CS_A_ERR_INVALID_WRITE_WAIT_STATE_VAL	CS_A_ConfigureChipSelectFeatures	Not a valid write wait state selection
CS_A_ERR_INVALID_PORT_SIZE_VAL	CS_A_ConfigureChipSelectFeatures	Not a valid port size selection
CS_A_ERR_INVALID_READ_ONLY_VAL	CS_A_ConfigureChipSelectFeatures	Not a valid read only selection
CS_A_ERR_INVALID_SUPERVISOR_ONLY_VAL	CS_A_ConfigureChipSelectFeatures	Not a valid supervisor only selection
CS_A_ERR_INVALID_CHIP_SELECT_ENABLE_VAL	CS_A_ControlChipSelect	Invalid chip select enable value
CS_A_ERR_INVALID_RESET_CONDITION_VAL	CS_A_Reset	Reset Condition is not valid

7.4 CS_A API Definitions

7.4.1 CS_A_Reset

7.4.1.1 Description

The CS_A_Reset function resets the chip select module to the default hardware values. Parameter checking is a compile-time option.

7.4.1.2 Prototype

```
CS_A_ReturnCode_t CS_A_Reset (
          pCS_A_t                   CSPtr,
          CS_A_ResetCondition_t     ResetConditionVal
)
```

7.4.1.3 Arguments

Data Type	Formal Name	Description	Valid Values
pCS_A_t	CSPtr	Chip Select module base address associated with this driver.	Any non-zero core processor data address.
CS_A_ResetCondition_t	ResetConditionVal	Determines Reset Conditions.	CS_A_EMULATION_MODE CS_A_16_BIT_PORT_SIZE_ENABLE CS_A_32_BIT_PORT_SIZE_ENABLE

7.4.1.4 Return Values

Error Code	Description
CS_A_ERR_NONE	No error
CS_A_ERR_INVALID_HANDLE	Chip Select module base address parameter is zero.
CS_A_ERR_INVALID_RESET_CONDITION_VAL	Reset Condition is not valid.

7.4.2 CS_A_ConfigureChipSelectFeatures

7.4.2.1 Description

The CS_A_ConfigureChipSelectFeatures function writes unformatted data to a chip select register with the user defined values requested to perform a chip select.

This function should only be called after the Chip Select (CS) module has been reset through a previous call to the CS_A_Reset function.

7.4.2.2 Prototype

```
CS_A_ReturnCode_t CS_A_ConfigureChipSelectFeatures (
         pCS_A_t                      CSPtr,
         CS_A_RegisterSwitch_t        CSRegSwitch,
         CS_A_TransferAckControl_t    TransferAckVal,
         CS_A_WaitStates_t            WaitStateVal,
         CS_A_WriteEnable_t           WriteEnableVal,
         CS_A_WriteWaitState_t        WriteWaitStateVal,
         CS_A_PortSize_t              PortSizeVal,
         CS_A_ReadOnly_t              ReadOnlyVal,
         CS_A_SupervisorOnly_t        SupervisorOnlyVal
)
```

7.4.2.3 Arguments

Data Type	Formal Name	Description	Valid Values
pCS_A_t	CSPtr	CS base address associated with this driver.	Any non-zero core processor data address.
CS_A_RegisterSwitch_t	CSRegSwitch	Select among CS_A registers.	CS_A_CSCR0 CS_A_ CSCR1 CS_A_ CSCR2 CS_A_CSCR3
CS_A_TransferAckControl_t	TransferAckVal	Transfer Acknowledge enabled or disabled.	CS_A_EXTERNAL_LOGIC_ASSERTION CS_A_CHIP_SELECT_LOGIC_ASSERTION
CS_A_WaitStates_t	WaitStateVal	Number of wait states for the chip select logic to insert before asserting the internal cycle termination signal.	CS_A_WAIT_STATE_ZERO CS_A_WAIT_STATE_ONE CS_A_WAIT_STATE_TWO CS_A_WAIT_STATE_THREE CS_A_WAIT_STATE_FOUR CS_A_WAIT_STATE_FIVE CS_A_WAIT_STATE_SIX
CS_A_WriteEnable_t	WriteEnableVal	Determines when the Enable Byte output pins are asserted.	CS_A_WRITE_AS_BYTES_ENABLE CS_A_EXTERNAL_WRITE_ENABLE
CS_A_WriteWaitState_t	WriteWaitStateVal	Determines additional wait state for write cycle.	CS_A_NO_WAIT_STATE_ADD CS_A_ADD_ONE_WAIT_STATE
CS_A_PortSize_t	PortSizeVal	Width of external data port.	CS_A_PORT_SIZE_16 CS_A_PORT_SIZE_32
CS_A_ReadOnly_t	ReadOnlyVal	Chip Select logic Read Only.	CS_A_READ_WRITE_ENABLE CS_A_READ_ONLY_ENABLE
CS_A_SupervisorOnly_t	SupervisorOnlyVal	Determines user level access restriction.	CS_A_SUPERVISOR_AND_USER_ENABLE CS_A_SUPERVISOR_ONLY_ENABLE

7.4.2.4 Return Values

Error Code	Description
CS_A_ERR_NONE	No error
CS_A_ERR_INVALID_HANDLE	CS base address parameter is zero
CS_A_ERR_INVALID_TRANSFER_ACK_VAL	Not a valid transfer acknowledge selection
CS_A_ERR_INVALID_WRITE_ENABLE_VAL	Not a valid write enable selection
CS_A_ERR_INVALID_WAIT_STATE_VAL	Not a valid wait state selection
CS_A_ERR_INVALID_WRITE_WAIT_STATE_VAL	Not a valid write wait state selection
CS_A_ERR_INVALID_PORT_SIZE_VAL	Not a valid port size selection
CS_A_ERR_INVALID_READ_ONLY_VAL	Not a valid read only selection
CS_A_ERR_INVALID_SUPERVISOR_ONLY_VAL	Not a valid supervisor only selection
CS_A_ERR_INVALID_REGISTER	CS Register Selection switch is invalid

7.4.3 CS_A_ControlChipSelect

7.4.3.1 Description

The CS_A_ControlChipSelect function enables and disables features of the Chip Select (CS) module. Parameter checking is a compile time option.

This function should only be called after the Chip Select (CS) module has been reset through a previous call to the CS_A_Reset function.

7.4.3.2 Prototype

```
CS_A_ReturnCode_t CS_A_ControlChipSelect (
          pCS_A_t                       CSPtr,
          CS_A_RegisterSwitch_t         CSRegSwitch,
          CS_A_ChipSelectControl_t      ChipSelectEnableVal
)
```

7.4.3.3 Arguments

Data Type	Formal Name	Description	Valid Values
pCS_A_t	CSPtr	CS base address associated with this driver.	Any non-zero core processor data address.
CS_A_RegisterSwitch_t	CSRegSwitch	Specifies a CS_A register.	CS_A_CSCR0 CS_A_ CSCR1 CS_A_ CSCR2 CS_A_CSCR3
CS_A_ChipSelectControl_t	ChipSelectEnableVal	Specifies whether the Chip Select module feature (CSRegSwitch) should be enabled or disabled.	CS_A_CHIP_SELECT_ENABLE CS_A_CHIP_SELECT_DISABLE

7.4.3.4 Return Values

Error Code	Description
CS_A_ERR_NONE	No error
CS_A_ERR_INVALID_HANDLE	CS base address parameter is zero
CS_A_ERR_INVALID_CHIP_SELECT_ENABLE_VAL	Invalid chip select enable value
CS_A_ERR_INVALID_REGISTER	CS Register Selection switch is invalid

7.4.4 CS_A_SetRegister

7.4.4.1 Description

The CS_A_SetRegister function writes unformatted data to selected Chip Select (CS) module registers. Parameter checking is a compile-time option.

This function can be called at any time.

7.4.4.2 Prototype

```
CS_A_ReturnCode_t CS_A_SetRegister (
         pCS_A_t                  CSPtr,
         CS_A_RegisterSwitch_t    CSRegSwitch,
         UINT16                   RegisterValue
)
```

7.4.4.3 Arguments

Data Type	Formal Name	Description	Valid Values
pCS_A_t	CSPtr	CS_A base address associated with this driver.	Any non-zero core processor data address.
CS_A_RegisterSwitch_t	CSRegSwitch	Select among CS_A registers.	CS_A_CSCR0 CS_A_ CSCR1 CS_A_ CSCR2 CS_A_CSCR3
UINT16	RegisterValue	Data to copy into selected CS_A register.	Any 16-bit value.

7.4.4.4 Return Values

Error Code	Description
CS_A_ERR_NONE	No error
CS_A_ERR_INVALID_HANDLE	CS base address parameter is zero
CS_A_ERR_INVALID_REGISTER	CS Register Selection switch is invalid

7.4.5 CS_A_GetRegister

7.4.5.1 Description

The CS_A_GetRegister function returns unformatted data from selected Chip Select (CS) module registers. Parameter checking is a compile-time option.

This function can be called at any time.

7.4.5.2 Prototype

```
CS_A_ReturnCode_t CS_A_GetRegister (
         pCS_A_t                    CSPtr,
         CS_A_RegisterSwitch_t      CSRegisterSwitch,
         UINT16                    *GetRegisterPtr
)
```

7.4.5.3 Arguments

Data Type	Formal Name	Description	Valid Values
pCS_A_t	CSPtr	CS_A base address associated with this driver.	Any non-zero core processor data address.
CS_A_RegisterSwitch_t	CSRegSwitch	Specifies a CS_A register.	CS_A_CSCR0 CS_A_ CSCR1 CS_A_ CSCR2 CS_A_CSCR3
UINT16 *	GetRegisterPtr	Address where the data from the CS_A register (specified by CSRegSwitch) will be written.	Any non-zero core processor data address.

7.4.5.4 Return Values

Error Code	Description
CS_A_ERR_NONE	No error
CS_A_ERR_INVALID_HANDLE	CS base address parameter is zero
CS_A_ERR_BAD_RESULT_ADDR	Result Pointer is zero
CS_A_ERR_INVALID_REGISTER	CS Register Selection switch is invalid

7.5 CS_A Example Application

The following describes the CS_A example application.

7.5.1 Description for Example Application

This program demonstrates features of the chip select register module. It will first reset the module, then show how to configure features, and finally enable the same features.

7.5.1.1 Example Application Code

```
/*********************************************************************

File:    csa_app.c

Purpose: The following example code illustrates the use of the M-CORE
         Peripherals Library CS_A Level 1 driver.

         This program demonstrates features of the chip select
         register module.  It will show how to configure features
         on two chip selects, and finally enable the same features.

         If an error occurs, the program will exit with the line
         number of where the failure occurred.

Time to run:
        Less than 30 seconds.

General Setup:
         General hardware and software setup to run this application
         can be found in the 'readme_app.txt' file, located in the
         '<library>/app' directory.

Application Specific Setup:
         Development Board: MMC2107

(C) Copyright Metrowerks, 2000.  All rights reserved.
*********************************************************************/

#include "cs_a.h"
#include "plibdefs_mmc2107.h"

/* macro to handle error checking and response */
#define HANDLE_ERROR(err)   if (err != CS_A_ERR_NONE) return(__LINE__);

int main(void);              /* function prototype */
int cs_example(pCS_A_t);     /* function prototype */
```

CS_A Level 1

```
/*********************************************************************
   Name: main
   Description: Drive program operation
   Inputs: csptr (pointer to CS_A base address)
   Outputs: none
*********************************************************************/
int main ()
{
    int status;

status = cs_example ((pCS_A_t)__MMC2107_CS);

    return (status);
}

/*********************************************************************
   Name: cs_example
   Description: Provide an example for the usage of the CS_A
   Inputs:  CSPtr    (pointer to CS_A base address)
   Outputs: status (if passed, 0; else line number of the failure)
*********************************************************************/
int cs_example (pCS_A_t CSPtr)
{
CS_A_ReturnCode_t retval;
retval = CS_A_ConfigureChipSelectFeatures (CSPtr,
                              CS_A_CSCR3,
                              CS_A_CHIP_SELECT_LOGIC_ASSERTION,
                              CS_A_WAIT_STATE_SIX,
                              CS_A_EXTERNAL_WRITE_ENABLE,
                              CS_A_ADD_ONE_WAIT_STATE,
                              CS_A_PORT_SIZE_32,
                              CS_A_READ_ONLY_ENABLE,
                              CS_A_SUPERVISOR_ONLY_ENABLE
                             );

HANDLE_ERROR(retval);

retval = CS_A_ControlChipSelect (CSPtr,            /* base address */
                         CS_A_CSCR3,              /* register */
                         CS_A_CHIP_SELECT_DISABLE  /* chip select enable */
                        );
HANDLE_ERROR(retval);

retval = CS_A_ConfigureChipSelectFeatures (CSPtr,
                              CS_A_CSCR0,
                              CS_A_EXTERNAL_LOGIC_ASSERTION,
                              CS_A_WAIT_STATE_FOUR,
                              CS_A_WRITE_AS_BYTES_ENABLE,
                              CS_A_NO_WAIT_STATE_ADD,
                              CS_A_PORT_SIZE_32,
                              CS_A_READ_WRITE_ENABLE,
                              CS_A_SUPERVISOR_AND_USER_ENABLE
                             );
HANDLE_ERROR(retval);
```

```
retval = CS_A_ControlChipSelect (CSPtr,
                        CS_A_CSCR0,                /* status structure */
                        CS_A_CHIP_SELECT_ENABLE    /* chip select enable */
                       );
HANDLE_ERROR(retval);
  return(0);

} /* end of main */
```

CS_A Level 1

Section 8 EdgePort_B Level 1

This section describes the EdgePort_B Level 1 M•CORE™ Device Driver.

8.1 Overview

The Edge Port and its accompanying driver (EdgePort_B) are used on the MMC2107 development system. The level 1 service is designed to allow the programmer to control the Edge Port on a near-register level, but without the concern of exact positions of registers and fields, or the detailed sequences of register operations. Refer to the MMC2107 Advance Information (Data Book) (MMC2107/D) for additional information.

8.2 EdgePort_B Service Functions

The EdgePort_B device driver software performs the following functions:

- Writes a 16-bit value to one of the Edge Port Registers.
- Gets the contents of one of the Edge Port Registers.
- Gets the Edge Port status.
- Enables or disables an Edge Port Pin.
- Sets data direction for an Edge Port Pin.
- Gets EdgePort pin states.
- Sets all EdgePort registers to their reset (power on) values.
- Writes to an Edge Port Register.
- Configures each of the Edge Port Pin Assignment interrupt pins.

8.3 EdgePort_B Data Type Definitions

The following paragraphs describe the data definitions recognized by the EdgePort_B device driver. These data definitions are grouped into six categories: standard data types, board specific base address definitions, derived data types, enumerated data types, register macros, and return codes.

8.3.1 Standard Data Type Definitions

- UINT8 — 8-bit unsigned data value
- INT8 — 8-bit signed data value
- UINT16 — 16-bit unsigned data value
- INT16 — 16-bit signed data value

- UINT32 — 32-bit unsigned data value
- INT32 — 32-bit signed data value

8.3.2 Board Specific Base Address Definitions

```
#define __MMC2107_EDGEPORT  (0x00C60000)
```

8.3.3 Derived Data Type Definitions

8.3.3.1 EdgePort_B_t - Edge Port Register Definition

```
typedef struct {
            UINT16    EPPAR;  /* Edge Port Pins Assignment Register */
            UINT8     EPDDR;  /* Edge Port Data Direction Register  */
            UINT8     EPFER;  /* Edge Port Flag Enable Register     */
   Volatile UINT8     EPDR;   /* Edge Port Data Register            */
   Volatile UINT8     EPPDR;  /* Edge Port Pin Data Register        */
   Volatile UINT8     EPFR;   /* Edge Port Flag Register            */
} EdgePort_B_t, *pEdgePort_B_t;
```

8.3.3.2 EdgePort_B_Status_t – Edge Port Status Structure Definition

```
typedef struct {
          EdgePort_B_Detect_t    Edge_Port_Flag_0;
          EdgePort_B_Detect_t    Edge_Port_Flag_1;
          EdgePort_B_Detect_t    Edge_Port_Flag_2;
          EdgePort_B_Detect_t    Edge_Port_Flag_3;
          EdgePort_B_Detect_t    Edge_Port_Flag_4;
          EdgePort_B_Detect_t    Edge_Port_Flag_5;
          EdgePort_B_Detect_t    Edge_Port_Flag_6;
          EdgePort_B_Detect_t    Edge_Port_Flag_7;
} EdgePort_B_Status_t, *pEdgePort_B_Status_t;
```

8.3.4 Enumerated Data Type Definitions

8.3.4.1 EdgePort_B_Register_t

```
typedef enum {
    EdgePort_B_EPPAR,     /* Select Edge Port Pin Assignment Register */
    EdgePort_B_EPDDR,     /* Select Edge Port Data Direction Register */
    EdgePort_B_EPFER,     /* Select Edge Port Flag Enable Register    */
    EdgePort_B_EPDR,      /* Select Edge Port Data Register           */
    EdgePort_B_EPPDR,     /*  Select Edge Port Pin Data Register      */
    EdgePort_B_EPFR       /* Select Edge Port Flag Register           */
} EdgePort_B_Register_t;
```

8.3.4.2 EdgePort_Detect_t – Edge Detected Selection

```
typedef enum {
    EdgePort_B_Pin_Not_Detected,   /* Pin change was not detected */
    EdgePort_B_Pin_Detected        /* Pin change was detected     */
} EdgePort_B_Detect_t;
```

8.3.4.3 EdgePort_B_Control_t - Control Selection

```
typedef enum {
     EdgePort_B_DISABLE,     /* Disable Selection */
     EdgePort_B_ENABLE       /* Enable Selection  */
} EdgePort_B_Control_t;
```

8.3.4.4 EdgePort_B_DataDir_t - Edge Port Pin Data Direction

```
typedef enum {
    EdgePort_B_DATADIR_INPUT,    /* Configure pin as input  */
    EdgePort_B_DATADIR_OUTPUT    /* Configure pin as output */
} EdgePort_B_DataDir_t;
```

8.3.4.5 EdgePort_B_Port_t - Edge Port Pin Data Ports

```
typedef enum {
    EdgePort_B_PortPin_0,     /* Select Edge Port Pin 0  */
    EdgePort_B_PortPin_1,     /* Select Edge Port Pin 1  */
    EdgePort_B_PortPin_2,     /* Select Edge Port Pin 2  */
    EdgePort_B_PortPin_3,     /* Select Edge Port Pin 3  */
    EdgePort_B_PortPin_4,     /* Select Edge Port Pin 4  */
    EdgePort_B_PortPin_5,     /* Select Edge Port Pin 5  */
    EdgePort_B_PortPin_6,     /* Select Edge Port Pin 6  */
    EdgePort_B_PortPin_7      /* Select Edge Port Pin 7  */
} EdgePort_B_Port_t;
```

8.3.4.6 EdgePort_B_PinType_t - Edge Port Detection Types

```
typedef enum {
    EdgePort_B_LevelDetect,     /* Level Sensitive Detection              */
    EdgePort_B_RisingEdge,      /* Rising Edge Detection                  */
    EdgePort_B_FallingEdge,     /* Falling Edge Detection                 */
    EdgePort_B_BothEdge         /* Both Falling and Rising Edge Detection */
} EdgePort_B_PinType_t;
```

8.3.4.7 EdgePort_B_DataVal_t - Edge Port Pin Read Value

```
typedef enum {
    EdgePort_B_CLEAR,      /* Data Value will be cleared with 0 */
    EdgePort_B_SET         /* Data Value will be set with 1     */
} EdgePort_B_DataVal_t;
```

8.3.5 Register Macros

8.3.5.1 Edge Port Pin Assignment Register Macro

```
#define EPPAR_EPPA0_BITNO  0
#define EPPAR_EPPA0_MASK   (2 << EPPAR_EPPA0_BITNO) /* 00 00 00 00 00 00 00 11 */
#define EPPAR_EPPA1_BITNO  2
#define EPPAR_EPPA1_MASK   (2 << EPPAR_EPPA1_BITNO) /* 00 00 00 00 00 00 11 00 */
#define EPPAR_EPPA2_BITNO  4
#define EPPAR_EPPA2_MASK   (2 << EPPAR_EPPA2_BITNO) /* 00 00 00 00 00 11 00 00 */
#define EPPAR_EPPA3_BITNO  6
#define EPPAR_EPPA3_MASK   (2 << EPPAR_EPPA3_BITNO) /* 00 00 00 00 11 00 00 00 */
#define EPPAR_EPPA4_BITNO  8
#define EPPAR_EPPA4_MASK   (2 << EPPAR_EPPA4_BITNO) /* 00 00 00 11 00 00 00 00 */
#define EPPAR_EPPA5_BITNO  10
#define EPPAR_EPPA5_MASK   (2 << EPPAR_EPPA5_BITNO) /* 00 00 11 00 00 00 00 00 */
#define EPPAR_EPPA6_BITNO  12
#define EPPAR_EPPA6_MASK   (2 << EPPAR_EPPA6_BITNO) /* 00 11 00 00 00 00 00 00 */
#define EPPAR_EPPA7_BITNO  14
#define EPPAR_EPPA7_MASK   (2 << EPPAR_EPPA7_BITNO) /* 11 00 00 00 00 00 00 00 */
#define EPPAR_EPPA_BITNO  0
#define EPPAR_EPPA_MASK   (0XFFFF)
#define EPPAR_RESET_MASK (0X00)
```

8.3.5.2 Edge Port Data Direction Register Macro

```
#define EPDDR_EPDD0_BITNO  0
#define EPDDR_EPDD0_MASK   (1 << EPDDR_EPDD0_BITNO) /* 0000 0001 */
#define EPDDR_EPDD1_BITNO  1
#define EPDDR_EPDD1_MASK   (1 << EPDDR_EPDD1_BITNO) /* 0000 0010 */
#define EPDDR_EPDD2_BITNO  2
#define EPDDR_EPDD2_MASK   (1 << EPDDR_EPDD2_BITNO) /* 0000 0100 */
#define EPDDR_EPDD3_BITNO  3
#define EPDDR_EPDD3_MASK   (1 << EPDDR_EPDD3_BITNO) /* 0000 1000 */
#define EPDDR_EPDD4_BITNO  4
#define EPDDR_EPDD4_MASK   (1 << EPDDR_EPDD4_BITNO) /* 0001 0000 */
#define EPDDR_EPDD5_BITNO  5
#define EPDDR_EPDD5_MASK   (1 << EPDDR_EPDD5_BITNO) /* 0010 0000 */
#define EPDDR_EPDD6_BITNO  6
#define EPDDR_EPDD6_MASK   (1 << EPDDR_EPDD6_BITNO) /* 0100 0000 */
#define EPDDR_EPDD7_BITNO  7
#define EPDDR_EPDD7_MASK   (1 << EPDDR_EPDD7_BITNO) /* 1000 0000 */
#define EPDDR_EPDD_BITNO   0
#define  EPDDR_EPDD_MASK (0XFF)
#define EPDDR_RESET_MASK   (0X00)
```

8.3.5.3 Edge Port Flag Enable Register

```
#define EPFER_EPFE0_BITNO 0
#define EPFER_EPFE0_MASK   (1 << EPFER_EPFE0_BITNO) /* 0000 0001 */
#define EPFER_EPFE1_BITNO 1
#define EPFER_EPFE1_MASK   (1 << EPFER_EPFE1_BITNO) /* 0000 0010 */
#define EPFER_EPFE2_BITNO 2
#define EPFER_EPFE2_MASK   (1 << EPFER_EPFE2_BITNO) /* 0000 0100 */
#define EPFER_EPFE3_BITNO 3
#define EPFER_EPFE3_MASK   (1 << EPFER_EPFE3_BITNO) /* 0000 1000 */
#define EPFER_EPFE4_BITNO 4
#define EPFER_EPFE4_MASK   (1 << EPFER_EPFE4_BITNO) /* 0001 0000 */
#define EPFER_EPFE5_BITNO 5
#define EPFER_EPFE5_MASK   (1 << EPFER_EPFE5_BITNO) /* 0010 0000 */
#define EPFER_EPFE6_BITNO 6
#define EPFER_EPFE6_MASK   (1 << EPFER_EPFE6_BITNO) /* 0100 0000 */
#define EPFER_EPFE7_BITNO 7
#define EPFER_EPFE7_MASK   (1 << EPFER_EPFE7_BITNO) /* 1000 0000 */
#define EPFER_EPFE_BITNO  0
#define  EPFER_EPFE_MASK (0XFF)
#define EPFER_RESET_MASK   (0X00)
```

8.3.5.4 Edge Port Data Register Macro

```
#define EPDR_EPD0_BITNO 0
#define EPDR_EPD0_MASK  (1 << EPDR_EPD0_BITNO) /* 0000 0001 */
#define EPDR_EPD1_BITNO 1
#define EPDR_EPD1_MASK  (1 << EPDR_EPD1_BITNO)      /* 0000 0010 */
#define EPDR_EPD2_BITNO 2
#define EPDR_EPD2_MASK  (1 << EPDR_EPD2_BITNO) /* 0000 0100 */
#define EPDR_EPD3_BITNO 3
#define EPDR_EPD3_MASK  (1 << EPDR_EPD3_BITNO) /* 0000 1000 */
#define EPDR_EPD4_BITNO 4
#define EPDR_EPD4_MASK  (1 << EPDR_EPD4_BITNO) /* 0001 0000 */
#define EPDR_EPD5_BITNO 5
#define EPDR_EPD5_MASK  (1 << EPDR_EPD5_BITNO) /* 0010 0000 */
#define EPDR_EPD6_BITNO 6
#define EPDR_EPD6_MASK  (1 << EPDR_EPD6_BITNO) /* 0100 0000 */
#define EPDR_EPD7_BITNO 7
#define EPDR_EPD7_MASK  (1 << EPDR_EPD7_BITNO) /* 1000 0000 */
#define EPDR_EPD_BITNO  0
#define  EPDR_EPDR_MASK (0XFF)
#define EPDR_RESET_MASK   (0XFF)
```

8.3.5.5 Edge Port Pin Data Register Macro

```
#define EPPDR_EPPD0_BITNO  0
#define EPPDR_EPPD0_MASK   (1 << EPPDR_EPPD0_BITNO) /* 0000 0001 */
#define EPPDR_EPPD1_BITNO  1
#define EPPDR_EPPD1_MASK   (1 << EPPDR_EPPD1_BITNO) /* 0000 0010 */
#define EPPDR_EPPD2_BITNO  2
#define EPPDR_EPPD2_MASK   (1 << EPPDR_EPPD2_BITNO) /* 0000 0100 */
#define EPPDR_EPPD3_BITNO  3
#define EPPDR_EPPD3_MASK   (1 << EPPDR_EPPD3_BITNO) /* 0000 1000 */
#define EPPDR_EPPD4_BITNO  4
#define EPPDR_EPPD4_MASK   (1 << EPPDR_EPPD4_BITNO) /* 0001 0000 */
#define EPPDR_EPPD5_BITNO  5
#define EPPDR_EPPD5_MASK   (1 << EPPDR_EPPD5_BITNO) /* 0010 0000 */
#define EPPDR_EPPD6_BITNO  6
#define EPPDR_EPPD6_MASK   (1 << EPPDR_EPPD6_BITNO) /* 0100 0000 */
#define EPPDR_EPPD7_BITNO  7
#define EPPDR_EPPD7_MASK   (1 << EPPDR_EPPD7_BITNO) /* 1000 0000 */
```

8.3.5.6 Edge Port Flag Register Macro

```
#define EPFR_EPF0_BITNO 0
#define EPFR_EPF0_MASK  (1 << EPFR_EPF0_BITNO) /* 0000 0001 */
#define EPFR_EPF1_BITNO 1
#define EPFR_EPF1_MASK  (1 << EPFR_EPF1_BITNO) /* 0000 0010 */
#define EPFR_EPF2_BITNO 2
#define EPFR_EPF2_MASK  (1 << EPFR_EPF2_BITNO) /* 0000 0100 */
#define EPFR_EPF3_BITNO 3
#define EPFR_EPF3_MASK  (1 << EPFR_EPF3_BITNO) /* 0000 1000 */
#define EPFR_EPF4_BITNO 4
#define EPFR_EPF4_MASK  (1 << EPFR_EPF4_BITNO) /* 0001 0000 */
#define EPFR_EPF5_BITNO 5
#define EPFR_EPF5_MASK  (1 << EPFR_EPF5_BITNO) /* 0010 0000 */
#define EPFR_EPF6_BITNO 6
#define EPFR_EPF6_MASK  (1 << EPFR_EPF6_BITNO) /* 0100 0000 */
#define EPFR_EPF7_BITNO 7
#define EPFR_EPF7_MASK  (1 << EPFR_EPF7_BITNO) /* 1000 0000 */
#define EPFR_EPF_BITNO  0
#define  EPDR_EPDR_MASK (0XFF)
#define EPDR_RESET_MASK   (0X00)
```

8.3.6 Return Codes

Return codes from the enumerated type “EdgePort_B_ReturnCode_t”.

Return Code	Functions Returning	Description
EDGEPORT_B_ERR_NONE	All	No error.
EDGEPORT_B_ERR_INVALID_HANDLE	All	Edge Port base address parameter is zero.
EDGEPORT_B_ERR_BAD_RESULT_ADDR	GetRegister, GetStatus, ReadPortData	Result Pointer is zero.
EDGEPORT_B_ERR_INVALID_REGISTER	SetRegister, GetRegister	Edge Port Register Selection switch is invalid.
EDGEPORT_B_ERR_INVALID_CONTROL	ControlInterrupt	Control choice is not valid.
EDGEPORT_B_ERR_INVALID_DATA_VALUE	SetRegister, WritePortData	16-bit value is greater than OXFF
EDGEPORT_B_ERR_INVALID_DATA_DIRECTION	ConfigureDataDirection	Port Data Direction is invalid.
EDGEPORT_B_ERR_INVALID_PIN_NUMBER	ControlInterrupt, ConfigureDataDirection, ReadPortData, WritePortData, ConfigurePortPin	Edge Port Pin Number is invalid.
EDGEPORT_B_ERR_INVALID_PIN_TYPE	ConfigurePortPin	Edge Port type is invalid.
EDGEPORT_B_ERR_INVALID_WRITE_VALUE	WritePortData	Value written into pin is not one or zero.

8.4 EdgePort_B API Definitions

8.4.1 EdgePort_B_SetRegister

8.4.1.1 Description

The EdgePort_B_SetRegister function writes a 16-bit value to a specified Edge Portr register. Parameter checking is a compile-time option.

This function can be called at any time.

8.4.1.2 Prototype

```
EdgePort_B_ReturnCode_t  EdgePort_B_SetRegister (
          pEdgePort_B_t            EdgePortPtr,
          EdgePort_B_Register_t    EdgePortRegister,
          UINT16                   RegisterValue
)
```

8.4.1.3 Arguments

Data Type	Formal Name	Description	Valid Values
pEdgePort_B_t	EdgePortPtr	Edge Port base address associated with this driver	Any non-zero core processor data address.
EdgePort_B_Register_t	EdgePortRegister	Select among Edge Port Registers.	EdgePort_B_EPPAR, EdgePort_B_EPDDR, EdgePort_B_EPFER, EdgePort_B_EPDR, EdgePort_B_EPPDR, EdgePort_B_EPFR
UINT16	RegisterValue	Data to copy into selected register.	0X00 to OXFF except for EdgePort_B_EPPAR which will be from OX00 to OXFFFF

8.4.1.4 Return Values

Error Code	Description
EDGEPORT_B_ERR_NONE	No error
EDGEPORT_B_ERR_INVALID_HANDLE	Edge Port base parameter is zero.
EDGEPORT_B_ERR_INVALID_REGISTER	Edge Port Register Selection switch is invalid
EDGEPORT_B_ERR_INVALID_DATA_VALUE	16-bit value is greater than 0XFF.

8.4.2 EdgePort_B_GetRegister

8.4.2.1 Description

The EdgePort_B_GetRegister function returns the contents of a specified Edge Port register. Parameter checking is a compile-time option.

This function can be called at any time.

8.4.2.2 Prototype

```
EdgePort_B_ReturnCode_t EdgePort_B_GetRegister (
          pEdgePort_B_t          EdgePortPtr,
          EdgePort_B_Register_t  EdgePortRegister,
          UINT16                *GetRegisterPtr
)
```

8.4.2.3 Arguments

Data Type	Formal Name	Description	Valid Values
pEdgePort_B_t	EdgePortPtr	Edge Port base address associated with this driver	Any non-zero core processor data address.
EdgePort_B_PinRegister_t	EdgePortPinRegister	Select among Edge Port Registers.	EdgePort_B_EPPAR, EdgePort_B_EPDDR, EdgePort_B_EPFER, EdgePort_B_EPDR, EdgePort_B_EPPDR, EdgePort_B_EPFR
UINT16*	GetRegisterPtr	Result address for selected Edge Port Register	Any non-zero core processor data address.

8.4.2.4 Return Values

Error Code	Description
EDGEPORT_B_ERR_NONE	No error
EDGEPORT_B_ERR_INVALID_HANDLE	Edge Port base address parameter is zero.
EDGEPORT_B_ERR_BAD_RESULT_ADDR	Result Pointer is zero
EDGEPORT_B_ERR_INVALID_REGISTER	Edge Port Register Selection switch is invalid

8.4.3 EdgePort_B_GetStatus

8.4.3.1 Description

The EdgePort_B_GetStatus function returns the Edge Port status. Parameter checking is a compile-time option.

This function can be called at any time.

8.4.3.2 Prototype

```
EdgePort_B_ReturnCode_ t EdgePort_B_GetStatus (
          pEdgePort_B_t           EdgePortPtr,
          pEdgePort_B_Status_t    StatusPtr
)
```

8.4.3.3 Arguments

Data Type	Formal Name	Description	Valid Values
pEdgePort_B_t	EdgePortPtr	Edge Port base address associated with this driver.	Any non-zero core processor data address.
pEdgePort_B_Status_t	StatusPtr	Structure for status fields	Any non-zero core processor data address.

8.4.3.4 Return Values

Error Code	Description
EDGEPORT_B_ERR_NONE	No error
EDGEPORT_B_ERR_INVALID_HANDLE	Edge Port base address parameter is zero
EDGEPORT_B_ERR_BAD_RESULT_ADDR	Result pointer is zero

8.4.4 EdgePort_B_ControlInterrupt

8.4.4.1 Description

The EdgePort_B_ControlInterrupt function enables or disables an Edge Port Pin. Parameter checking is a compile time option.

This function can be called at any time.

8.4.4.2 Prototype

```
EdgePort_B_ReturnCode_t   EdgePort_B_ControlInterrupt (
          pEdgePort_B_t           EdgePortPtr,
          EdgePort_B_Port_t       EdgePortPin,
          EdgePort_B_Control_t    InterruptVal
)
```

8.4.4.3 Arguments

Data Type	Formal Name	Description	Valid Values
pEdgePort_B_t	EdgePortPtr	Edge Port base address associated with this driver.	Any non-zero core processor data address.
EdgePort_B_Port_t	EdgePortPin	Select Edge Port Pin.	EdgePort_B_PortPin_0, EdgePort_B_PortPin_1, EdgePort_B_PortPin_2, EdgePort_B_PortPin_3, EdgePort_B_PortPin_4, EdgePort_B_PortPin_5, EdgePort_B_PortPin_6, EdgePort_B_PortPin_7
EdgePort_B_Control_t	InterruptVal	Control Interrupt Requests	EdgePort_B_DISABLE, EdgePort_B_ENABLE

8.4.4.4 Return Values

Error Code	Description
EDGEPORT_B_ERR_NONE	No error
EDGEPORT_B_ERR_INVALID_HANDLE	Edge Port base address parameter is zero
EDGEPORT_B_ERR_INVALID_PIN_NUMBER	Edge Port Pin Number is invalid.
EDGEPORT _B_ERR_INVALID_CONTROL	Not a valid control selection

8.4.5 EdgePort_B_ConfigureDataDirection

8.4.5.1 Description

The EdgePort_B_ConfigureDataDirection function sets the data direction for an Edge Port Pin. Parameter checking is a compile-time option.

This function can be called at any time if used as General Purpose I/O. Otherwise, it should only be called after the Edge Port has been configured through a previous call to the EdgePort_B_ConfigurePortPin function.

8.4.5.2 Prototype

```
EdgePort_B_ReturnCode_t   EdgePort_B_ConfigureDataDirection (
          pEdgePort_B_t          EdgePortPtr,
          EdgePort_B_DataDir_t   DataDir,
          EdgePort_B_Port_t      EdgePortPin
)
```

8.4.5.3 Arguments

Data Type	Formal Name	Description	Valid Values
pEdgePort_B_t	EdgePortPtr	Edge Port base address associated with this driver.	Any non-zero core processor data address.
EdgePort_B_DataDir_t	DataDir	Configure Data Direction	EdgePort_B_DATADIR_INPUT, EdgePort_B_DATADIR_OUTPUT
EdgePort_B_Port_t	EdgePortPin	Select Edge Port Pin	EdgePort_B_PortPin_0, EdgePort_B_PortPin_1, EdgePort_B_PortPin_2, EdgePort_B_PortPin_3, EdgePort_B_PortPin_4, EdgePort_B_PortPin_5, EdgePort_B_PortPin_6, EdgePort_B_PortPin_7

8.4.5.4 Return Values

Error Code	Description
EDGEPORT_B_ERR_NONE	No error
EDGEPORT_B_ERR_INVALID_HANDLE	Edge Port base address parameter is zero
EDGEPORT_B_ERR_INVALID_DATA_DIRECTION	Port Data Direction is invalid.
EDGEPORT_B_ERR_INVALID_PIN_NUMBER	Data Direction Pin is invalid.

8.4.6 EdgePort_B_ReadPortData

8.4.6.1 Description

The EdgePort_B_ReadPortData function returns EdgePort pin states. Parameter checking is a compile-time option.

This function can be called at any time.

8.4.6.2 Prototype

```
EdgePort_B_ReturnCode_t   EdgePort_B_ReadPortData (
          pEdgePort_B_t           EdgePortPtr,
          EdgePort_B_Port_t       EdgePortPin,
          pEdgePort_B_DataVal_t   ReadPortPtr
)
```

8.4.6.3 Arguments

Data Type	Formal Name	Description	Valid Values
pEdgePort_B_t	EdgePortPtr	EdgePort_B base address associated with this driver.	Any non-zero core processor data address.
EdgePort_B_Port_t	EdgePortPin	Select EdgePort port	EdgePort_B_PortPin_0, EdgePort_B_PortPin_1, EdgePort_B_PortPin_2, EdgePort_B_PortPin_3, EdgePort_B_PortPin_4, EdgePort_B_PortPin_5, EdgePort_B_PortPin_6, EdgePort_B_PortPin_7
pEdgePort_B_DataVal_t	ReadPortPtr	Data to be copied into selected Edge Port Pin.	EdgePort_B_CLEAR, EdgePort_B_SET

8.4.6.4 Return Values

Error Code	Description
EDGEPORT_B_ERR_NONE	No error
EDGEPORT_B_ERR_INVALID_HANDLE	Edge Port base address parameter is zero.
EDGEPORT_B_ERR_INVALID_PIN_NUMBER	Port pin number is invalid.
EDGEPORT_B_ERR_BAD_RESULT_ADDRESS	Result Pointer is zero.

8.4.7 EdgePort_B_Reset

8.4.7.1 Description

The EdgePort_B_Reset function sets all EdgePort registers to their reset (power on) values. Parameter checking is a compile-time option.

This function can be called at any time.

8.4.7.2 Prototype

```
EdgePort_B_ReturnCode_t   EdgePort_B_Reset (
          pEdgePort_B_t    EdgePortPtr,
)
```

8.4.7.3 Arguments

Data Type	Formal Name	Description	Valid Values
pEdgePort_B_t	EdgePortPtr	EdgePort_B base address associated with this driver.	Any non-zero core processor data address.

8.4.7.4 Return Values

Error Code	Description
EDGEPORT_B_ERR_NONE	No error
EDGEPORT_B_ERR_INVALID_HANDLE	EdgePort base address parameter is zero

8.4.8 EdgePort_B_WritePortData

8.4.8.1 Description

The EdgePort_B_WritePortData function writes to an Edge Port Register. Parameter checking is a compile-time option.

This function can be called at any time.

8.4.8.2 Prototype

```
EdgePort_B_ReturnCode_t   EdgePort_B_ WritePortData (
          pEdgePort_B_t           EdgePortPtr,
          EdgePort_B_Port_t       EdgePortPin,
          EdgePort_B_DataVal_t    DataVal
)
```

8.4.8.3 Arguments

Data Type	Formal Name	Description	Valid Values
pEdgePort_B_t	EdgePortPtr	EdgePort_B base address associated with this driver.	Any non-zero core processor data address.
EdgePort_B_Port_t	EdgePortPin	Select Edge Port Pin	EdgePort_B_PortPin_0, EdgePort_B_PortPin_1, EdgePort_B_PortPin_2, EdgePort_B_PortPin_3, EdgePort_B_PortPin_4, EdgePort_B_PortPin_5, EdgePort_B_PortPin_6, EdgePort_B_PortPin_7
EdgePort_B_DataVal_t	DataVal	Value to write to the Edge Port Pin	EdgePort_B_CLEAR, EdgePort_B_SET

8.4.8.4 Return Values

Error Code	Description
EDGEPORT_B_ERR_NONE	No error
EDGEPORT_B_ERR_INVALID_HANDLE	Edge ort base address parameter is zero.
EDGEPORT_B_ERR_INVALID_DATA_VALUE	Value written into pin is not one or zero.
EDGEPORT_B_ERR_INVALID_PIN_NUMBER	Edge Port Switch is invalid.

8.4.9 EdgePort_B_ConfigurePortPin

8.4.9.1 Description

The EdgePort_B_ConfigurePortPin function configures each of the Edge Port Pin Assignment interrupt pins. Parameter checking is a compile-time option.

This function can be called at any time.

8.4.9.2 Prototype

```
EdgePort_B_ReturnCode_t   EdgePort_B_ConfigurePortPin (
          pEdgePort_B_t           EdgePortPtr,
          EdgePort_B_Port_t       EdgePortPin,
          EdgePort_B_PinType_t    PortType
)
```

8.4.9.3 Arguments

Data Type	Formal Name	Description	Valid Values
pEdgePort_B_t	EdgePortPtr	EdgePort_B base address associated with this driver.	Any non-zero core processor data address.
EdgePort_B_Port_t	EdgePortPin	Select Edge Port Pin.	EdgePort_B_PortPin_0, EdgePort_B_PortPin_1, EdgePort_B_PortPin_2, EdgePort_B_PortPin_3, EdgePort_B_PortPin_4, EdgePort_B_PortPin_5, EdgePort_B_PortPin_6, EdgePort_B_PortPin_7
EdgePort_B_PinType_t	EdgePortType	Value to write to the Edge Port Pin	EdgePort_B_LevelDetect, EdgePort_B_RisingEdge, EdgePort_B_FallingEdge, EdgePort_B_BothEdge

8.4.9.4 Return Values

Error Code	Description
EDGEPORT_B_ERR_NONE	No error
EDGEPORT_B_ERR_INVALID_HANDLE	Edge Port base address parameter is zero.
EDGEPORT_B_ERR_INVALID_PIN_NUMBER	Edge Port Pin Number is invalid.
EDGEPORT_B_ERR_INVALID_PIN_TYPE	Edge Port type is invalid.

8.5 EdgePort_B Example Application

The following describes the EdgePort_B example application.

8.5.1 Description for Example Application

The example application resets all bits of the Edge Port Pin Assignment Register. The program will configure pins EPPA7 as output. The program will then configure pin EPPA4 as rising edge triggered and as an input. The program will finally set pin EPPA7. NOTE: Pin 7 has to be configured as an output and pins[5:0] as the input. Pins 7 and 6 are Tsiz and can be run configured as either input or output at the same time. This application can only run in master mode. This program will continue running until either an edge was detected or it has exceeded the counter value.

8.5.1.1 Example Application Code

```
/*******************************************************************

File:    edgeport_app.c

Purpose: The following example code illustrates the use of the M-CORE
         Peripherals Library EdgePort_B Level 1 driver.  It is setup
         to operate on an M-CORE 2107 CMB board.

         The program resets all bits of the Edge Port Pin Assignment
         Register. The program will configure pins EPPA7 as output.
         The program will then configure pin EPPA4 as rising edge
         triggered and as an input.  The program will finally set
         pin EPPA7. NOTE: Pin 7 has to be configured as an output and
         pins[5:0] as the input.  Pins 7 and 6 are Tsiz and can be run
         configured as either input or output at the same time. This
         application can only run in master mode.

         This program will continue running until either an edge was
         detected or it has exceeded the counter value.

         If an error occurs, the program will exit with the line
         number of where the failure occurred.

Time to run:
         About 30 seconds.

General Setup:
         General hardware and software setup to run this application
         can be found in the 'readme_app.txt' file, located in the
         '<library>/app' directory.

Application Specific Setup:
         Development Board: MMC2107 CMB or EVB

(C) Copyright Motorola Inc, 2000.  All rights reserved.
*******************************************************************/
```

```
#include "edgeport_b.h"


/*****************************************************************/
/*  Macro Definitions                                            */
/*****************************************************************/
/* macro to handle error checking and response */
#define HANDLE_ERROR(err) if (err != EDGEPORT_B_ERR_NONE) return(__LINE__);
#ifndef __MMC2107_EDGEPORT
#define __MMC2107_EDGEPORT ((pEdgePort_B_t) (0x00C60000))
#endif
#undef EdgePort_B_PARAM_CHECKING       /* do in case of external definition */
#define EdgePort_B_PARAM_CHECKING 1    /* 1= enable for debug, 0 = disable  */

/*****************************************************************/
/*  Function Prototypes                                          */
/*****************************************************************/

int main(void);                            /* function prototype */
int edgeportb_example(pEdgePort_B_t);      /* function prototype */

/*****************************************************************/
/* Name: main                                                    */
/* Description: Drive program operation                          */
/* Inputs: EdgePortPtr(pointer to EdgePort_B base address)       */
/* Outputs: status (if passed, 0; else line number of the        */
/* failures)                                                     */
/*****************************************************************/
int main ()
{
    int status;

    status = edgeportb_example ((pEdgePort_B_t)__MMC2107_EDGEPORT);
    return (status);
}

/*****************************************************************/
/* Name: edgeportb_example                                       */
/* Description: Provide an example for the usage of the          */
/*              EdgePort_B.                                      */
/* Inputs:  EdgePortPtr (pointer to EdgePort_B base address)     */
/* Outputs: status (if passed, 0; else line number of the        */
/*          failure)                                             */
/*****************************************************************/
int edgeportb_example (pEdgePort_B_t EdgePortPtr)
{
/*---------------------------------------------------------------*/
/* declare local variables                                       */
/*---------------------------------------------------------------*/
/* return codes */
EdgePort_B_ReturnCode_t status=EDGEPORT_B_ERR_NONE; /*return value*/
UINT16 y;
EdgePort_B_Status_t Status;
```

EdgePort_B Level 1

```
pEdgePort_B_Status_t StatusPtr=&Status;
/* EdgePort_B_Port_t EdgePortPin;*/

EdgePort_B_PinType_t PortType;
EdgePort_B_PinType_t * PortTypePtr = &PortType;

EdgePort_B_Detect_t readyFlag = EdgePort_B_Pin_Not_Detected;

/*-----------------------------------------------------------------*/
/* resets the Edge Port                                            */
/*-----------------------------------------------------------------*/
/* Call EdgePort_B_Reset with valid parameters */
status = EdgePort_B_Reset(EdgePortPtr);

/* If an error code is returned, exit the program */
HANDLE_ERROR(status);

/*-----------------------------------------------------------------*/
/* Configure Port Pin 7 as an output.                              */
/*-----------------------------------------------------------------*/

status = EdgePort_B_ConfigureDataDirection(
      EdgePortPtr,                  /* Edge Port base address  */
      EdgePort_B_DATADIR_OUTPUT,     /* Port Pin 7 as output   */
      EdgePort_B_PortPin_7      /* enable Port Pin 7 as output  */
                                    );

/*-----------------------------------------------------------------*/
/* enable rising edge detection for Edge Port Pin 6                */
/*-----------------------------------------------------------------*/

status = EdgePort_B_ConfigurePortPin(
      EdgePortPtr,                  /* Edge Port base address  */
      EdgePort_B_PortPin_4,          /* enable Edge Port Pin 4 */
      EdgePort_B_RisingEdge     /* enable Rising Edge Detection */
                                    );

/*-----------------------------------------------------------------*/
/* enable Edge Port Pin 4 as a input.                              */
/*-----------------------------------------------------------------*/

status = EdgePort_B_ConfigureDataDirection(
      EdgePortPtr,                  /* Edge Port base address  */
      EdgePort_B_DATADIR_INPUT,   /* enable Port Pin 6 as input */
      EdgePort_B_PortPin_4                /* select port pin 4 */
                                    );
/*-----------------------------------------------------------------*/
/* inputtin a one into Edge Port Pin 7.                            */
/*-----------------------------------------------------------------*/

status = EdgePort_B_WritePortData(
      EdgePortPtr,                  /* Edge Port base address  */
      EdgePort_B_PortPin_7,               /* select Port Pin 7 */
```

```
      EdgePort_B_SET            /* inputting a one into Pin 7.  */
                                     );

/*---------------------------------------------------------------*/
/* Loop forever                                                  */
/*---------------------------------------------------------------*/
for(y=0; y<0xFFFF; y++)
{

/*---------------------------------------------------------------*/
/* Wait for for user to configure to pins                        */
/*---------------------------------------------------------------*/
   while (readyFlag == EdgePort_B_Pin_Not_Detected)
   {
   /* try to receive */
   status = EdgePort_B_GetStatus( EdgePortPtr, /* Edge Port base address */
                                  StatusPtr    /* structure for status  */
                                );
   /* If an error code is returned, exit the program */
   HANDLE_ERROR(status);

   /* check reception status */
   if (StatusPtr->Edge_Port_Flag_4 == EdgePort_B_Pin_Detected)
         {
            readyFlag = EdgePort_B_Pin_Detected;
            return(status);
         }
      } /* end while loop */
      readyFlag = EdgePort_B_Pin_Not_Detected;  /* reset readyFlag */
   }  /* end of for loop */
   return(status);
}  /* end of main */
```

Section 9 ITCN_B Level 1

This section describes the ITCN_B Level 1 M•CORE™ Device Driver.

9.1 Overview

The Interrupt Controller (ITCN) and its accompanying driver (ITCN_B) are used on the MMC2107 development system. The level 1 service is designed to allow the programmer to control the ITCN on a near-register level, but without the concern of exact positions of registers and fields, or the detailed sequences of register operations. Refer to the MMC2107 Advance Information (Data Book) (MMC2107/D) for additional information.

9.2 ITCN_B Service Functions

The ITCN_B device driver software provides these functions:

- Initializes the ITCN to a user-determined condition.
- Initalizes the vector table with user-supplied vector values for fast and normal interrupts.
- Sets the priority level for a selected interrupt source and sets a custom ISR value in the vector table for this interrupt source.
- Enables or disables the interrupts at a given priority level.
- Forces or clears and interrupt condition on the source specified.
- Gets the status of the ITCN.
- Gets unformatted data from selected ITCN registers.
- Writes unformatted data to ITCN registers.
- Resets all ITCN registers to their power-on reset state.

9.3 ITCN_B Data Type Definitions

The following paragraphs describe the data definitions recognized by the ITCN_B device driver. These data definitions are grouped into seven categories: standard data types, board specific base address definitions, derived data types, enumerated data types, register macros, miscellaneous macros, and return codes.

9.3.1 Standard Data Type Definitions

- UINT8 — 8-bit unsigned data value
- INT8 — 8-bit signed data value
- UINT16 — 16-bit unsigned data value

- INT16 — 16-bit signed data value
- UINT32 — 32-bit unsigned data value
- INT32 — 32-bit signed data value

9.3.2 Board Specific Base Address Definitions

```
#define __MMC2107_ITCN   (0x0000_C500)
```

9.3.3 Derived Data Type Definitions

9.3.3.1 ITCN_B_t - ITCN_B Register Definitions

```
typedef struct
{
          UINT16 ICR;         /* Interrupt Control Register           */
  volatile UINT16 ISR;        /* Interrupt Status Register            */
          UINT32 IFRH;        /* Interrupt Force Register High        */
          UINT32 IFRL;        /* Interrupt Force Register Low         */
  volatile UINT32 IPR;        /* Interrupt Pending Register           */
          UINT32 NIER;        /* Normal Interrupt Enable Register     */
  volatile UINT32 NIPR;       /* Normal Interrupt Pending Register    */
          UINT32 FIER;        /* Fast Interrupt Enable Register       */
  volatile UINT32 FIPR;       /* Fast Interrupt Pending Register      */
          UINT32 RESERVED1;   /* Reserved space, offset $20           */
          UINT32 RESERVED2;   /* Reserved space, offset $24           */
          UINT32 RESERVED3;   /* Reserved space, offset $28           */
          UINT32 RESERVED4;   /* Reserved space, offset $2C           */
          UINT32 RESERVED5;   /* Reserved space, offset $30           */
          UINT32 RESERVED6;   /* Reserved space, offset $34           */
          UINT32 RESERVED7;   /* Reserved space, offset $38           */
          UINT32 RESERVED8;   /* Reserved space, offset $3C           */
          UINT8  PLSR0;       /* Priority Level Select Register 0     */
          UINT8  PLSR1;       /* Priority Level Select Register 1     */
          UINT8  PLSR2;       /* Priority Level Select Register 2     */
          UINT8  PLSR3;       /* Priority Level Select Register 3     */
          UINT8  PLSR4;       /* Priority Level Select Register 4     */
          UINT8  PLSR5;       /* Priority Level Select Register 5     */
          UINT8  PLSR6;       /* Priority Level Select Register 6     */
          UINT8  PLSR7;       /* Priority Level Select Register 7     */
          UINT8  PLSR8;       /* Priority Level Select Register 8     */
          UINT8  PLSR9;       /* Priority Level Select Register 9     */
          UINT8  PLSR10;      /* Priority Level Select Register 10    */
          UINT8  PLSR11;      /* Priority Level Select Register 11    */
          UINT8  PLSR12;      /* Priority Level Select Register 12    */
          UINT8  PLSR13;      /* Priority Level Select Register 13    */
          UINT8  PLSR14;      /* Priority Level Select Register 14    */
          UINT8  PLSR15;      /* Priority Level Select Register 15    */
          UINT8  PLSR16;      /* Priority Level Select Register 16    */
          UINT8  PLSR17;      /* Priority Level Select Register 17    */
          UINT8  PLSR18;      /* Priority Level Select Register 18    */
          UINT8  PLSR19;      /* Priority Level Select Register 19    */
```

```
            UINT8  PLSR20;      /* Priority Level Select Register 20     */
            UINT8  PLSR21;      /* Priority Level Select Register 21     */
            UINT8  PLSR22;      /* Priority Level Select Register 22     */
            UINT8  PLSR23;      /* Priority Level Select Register 23     */
            UINT8  PLSR24;      /* Priority Level Select Register 24     */
            UINT8  PLSR25;      /* Priority Level Select Register 25     */
            UINT8  PLSR26;      /* Priority Level Select Register 26     */
            UINT8  PLSR27;      /* Priority Level Select Register 27     */
            UINT8  PLSR28;      /* Priority Level Select Register 28     */
            UINT8  PLSR29;      /* Priority Level Select Register 29     */
            UINT8  PLSR30;      /* Priority Level Select Register 30     */
            UINT8  PLSR31;      /* Priority Level Select Register 31     */
            UINT8  PLSR32;      /* Priority Level Select Register 32     */
            UINT8  PLSR33;      /* Priority Level Select Register 33     */
            UINT8  PLSR34;      /* Priority Level Select Register 34     */
            UINT8  PLSR35;      /* Priority Level Select Register 35     */
            UINT8  PLSR36;      /* Priority Level Select Register 36     */
            UINT8  PLSR37;      /* Priority Level Select Register 37     */
            UINT8  PLSR38;      /* Priority Level Select Register 38     */
            UINT8  PLSR39;      /* Priority Level Select Register 39     */
} ITCN_B_t, *pITCN_B_t;
```

9.3.3.2 ITCN_B_Status_t - Status Structure

```
typedef struct
{
    ITCN_B_Boolean_t normalInterruptAsserted;
    ITCN_B_Boolean_t fastInterruptAsserted;
    UINT8            vectorNumber;

} ITCN_B_Status_t, *pITCN_B_Status_t;
```

9.3.4 Enumerated Data Type Definitions

9.3.4.1 ITCN_B_Boolean_t – Boolean Enum

```
typedef enum
{
     ITCN_B_FALSE = 0,
     ITCN_B_TRUE
} ITCN_B_Boolean_t;
```

9.3.4.2 ITCN_B_ReturnCode_t – Error return codes for ITCN_B

```
typedef enum
{
  ITCN_B_ERR_NONE = 0x0L,                   /* No error (force long)      */
  ITCN_B_ERR_INVALID_HANDLE,                /* Invalid device handle      */
  ITCN_B_ERR_INVALID_VECTOR_TYPE,           /* Invalid vector type        */
  ITCN_B_ERR_INVALID_FAST_INTERRUPT_CONTROL, /* Invalid fast interrupt    */
                                            /*         control selection  */
  ITCN_B_ERR_INVALID_MASK_MODE,             /* Invalid mask mode          */
```

```
  ITCN_B_ERR_INVALID_MASK_LEVEL,            /* Invalid mask level         */
  ITCN_B_ERR_INVALID_PRIORITY_LEVEL,        /* Invalid priority level     */
  ITCN_B_ERR_INVALID_INTERRUPT_TYPE,        /* Invalid interrupt type     */
  ITCN_B_ERR_INVALID_ISR_ADDRESS,           /* Invalid ISR address        */
  ITCN_B_ERR_INVALID_INTERRUPT_SOURCE,      /* Invalid interrupt source   */
  ITCN_B_ERR_INVALID_INTERRUPT_CONTROL,     /* Invalid interrupt control  */
                                            /*         selection          */
  ITCN_B_ERR_INVALID_FORCE_CONTROL,         /* Invalid force control      */
                                            /*         selection          */
  ITCN_B_ERR_BAD_RESULT_ADDR,               /* bad result address         */
  ITCN_B_ERR_INVALID_REGISTER,              /* invalid register selection */
  ITCN_B_ERR_INVALID_REGISTER_VALUE         /* invalid register value     */
} ITCN_B_ReturnCode_t;          /* ITCN_B return codes */
```

9.3.4.3 ITCN_B_Register_t – INTC_B Register Selection

```
typedef enum
{
  ITCN_B_ICR,         /* Interrupt Control Register           */
  ITCN_B_ISR,         /* Interrupt Status Register            */
  ITCN_B_IFRH,        /* Interrupt Force Register High        */
  ITCN_B_IFRL,        /* Interrupt Force Register Low         */
  ITCN_B_IPR,         /* Interrupt Pending Register           */
  ITCN_B_NIER,        /* Normal Interrupt Enable Register     */
  ITCN_B_NIPR,        /* Normal Interrupt Pending Register    */
  ITCN_B_FIER,        /* Fast Interrupt Enable Register       */
  ITCN_B_FIPR,        /* Fast Interrupt Pending Register      */
  ITCN_B_PLSR0,       /* Priority Level Select Register 0     */
  ITCN_B_PLSR1,       /* Priority Level Select Register 1     */
  ITCN_B_PLSR2,       /* Priority Level Select Register 2     */
  ITCN_B_PLSR3,       /* Priority Level Select Register 3     */
  ITCN_B_PLSR4,       /* Priority Level Select Register 4     */
  ITCN_B_PLSR5,       /* Priority Level Select Register 5     */
  ITCN_B_PLSR6,       /* Priority Level Select Register 6     */
  ITCN_B_PLSR7,       /* Priority Level Select Register 7     */
  ITCN_B_PLSR8,       /* Priority Level Select Register 8     */
  ITCN_B_PLSR9,       /* Priority Level Select Register 9     */
  ITCN_B_PLSR10,      /* Priority Level Select Register 10    */
  ITCN_B_PLSR11,      /* Priority Level Select Register 11    */
  ITCN_B_PLSR12,      /* Priority Level Select Register 12    */
  ITCN_B_PLSR13,      /* Priority Level Select Register 13    */
  ITCN_B_PLSR14,      /* Priority Level Select Register 14    */
  ITCN_B_PLSR15,      /* Priority Level Select Register 15    */
  ITCN_B_PLSR16,      /* Priority Level Select Register 16    */
  ITCN_B_PLSR17,      /* Priority Level Select Register 17    */
  ITCN_B_PLSR18,      /* Priority Level Select Register 18    */
  ITCN_B_PLSR19,      /* Priority Level Select Register 19    */
  ITCN_B_PLSR20,      /* Priority Level Select Register 20    */
  ITCN_B_PLSR21,      /* Priority Level Select Register 21    */
  ITCN_B_PLSR22,      /* Priority Level Select Register 22    */
  ITCN_B_PLSR23,      /* Priority Level Select Register 23    */
  ITCN_B_PLSR24,      /* Priority Level Select Register 24    */
  ITCN_B_PLSR25,      /* Priority Level Select Register 25    */
```

```
  ITCN_B_PLSR26,      /* Priority Level Select Register 26     */
  ITCN_B_PLSR27,      /* Priority Level Select Register 27     */
  ITCN_B_PLSR28,      /* Priority Level Select Register 28     */
  ITCN_B_PLSR29,      /* Priority Level Select Register 29     */
  ITCN_B_PLSR30,      /* Priority Level Select Register 30     */
  ITCN_B_PLSR31,      /* Priority Level Select Register 31     */
  ITCN_B_PLSR32,      /* Priority Level Select Register 32     */
  ITCN_B_PLSR33,      /* Priority Level Select Register 33     */
  ITCN_B_PLSR34,      /* Priority Level Select Register 34     */
  ITCN_B_PLSR35,      /* Priority Level Select Register 35     */
  ITCN_B_PLSR36,      /* Priority Level Select Register 36     */
  ITCN_B_PLSR37,      /* Priority Level Select Register 37     */
  ITCN_B_PLSR38,      /* Priority Level Select Register 38     */
  ITCN_B_PLSR39       /* Priority Level Select Register 39     */
} ITCN_B_Register_t;
```

9.3.4.4 ITCN_B_Autovector_t – Specify autovectored or vectored interrupts

```
typedef enum
{
  ITCN_B_VECTORED_INTERRUPTS,
  ITCN_B_AUTOVECTORED_INTERRUPTS
} ITCN_B_Autovector_t;
```

9.3.4.5 ITCN_B_FastInterruptControl_t – Enable or disable fast interrupts

```
typedef enum
{
  ITCN_B_FAST_INTERRUPT_DISABLE,
  ITCN_B_FAST_INTERRUPT_ENABLE
} ITCN_B_FastInterruptControl_t;
```

9.3.4.6 ITCN_B_Mask_t – Specify Mask Mode

```
typedef enum
{
  ITCN_B_MASK_DISABLE,
  ITCN_B_MASK_ENABLE_ONLY_NORMAL,
  ITCN_B_MASK_ENABLE_FAST_AND_NORMAL
} ITCN_B_Mask_t;
```

9.3.4.7 ITCN_B_InterruptType_t – Specify fast or normal interrupts

```
typedef enum
{
  ITCN_B_INTERRUPT_NORMAL,
  ITCN_B_INTERRUPT_FAST
} ITCN_B_InterruptType_t;
```

9.3.4.8 ITCN_B_InterruptSource_t – Specify the source of the interrupt

```
typedef enum
{
```

```
  ITCN_B_ADC_PF1_SOURCE_0 = 0,
  ITCN_B_ADC_CF1_SOURCE_1,
  ITCN_B_ADC_PF2_SOURCE_2,
  ITCN_B_ADC_CF2_SOURCE_3,
  ITCN_B_SPI_MODF_SOURCE_4,
  ITCN_B_SPI_SPIF_SOURCE_5,
  ITCN_B_SCI1_TDRE_SOURCE_6,
  ITCN_B_SCI1_TC_SOURCE_7,
  ITCN_B_SCI1_RDRF_SOURCE_8,
  ITCN_B_SCI1_OR_SOURCE_9,
  ITCN_B_SCI1_IDLE_SOURCE_10,
  ITCN_B_SCI2_TDRE_SOURCE_11,
  ITCN_B_SCI2_TC_SOURCE_12,
  ITCN_B_SCI2_RDRF_SOURCE_13,
  ITCN_B_SCI2_OR_SOURCE_14,
  ITCN_B_SCI2_IDLE_SOURCE_15,
  ITCN_B_TIM1_C0F_SOURCE_16,
  ITCN_B_TIM1_C1F_SOURCE_17,
  ITCN_B_TIM1_C2F_SOURCE_18,
  ITCN_B_TIM1_C3F_SOURCE_19,
  ITCN_B_TIM1_TOF_SOURCE_20,
  ITCN_B_TIM1_PAIF_SOURCE_21,
  ITCN_B_TIM1_PAOVF_SOURCE_22,
  ITCN_B_TIM2_C0F_SOURCE_23,
  ITCN_B_TIM2_C1F_SOURCE_24,
  ITCN_B_TIM2_C2F_SOURCE_25,
  ITCN_B_TIM2_C3F_SOURCE_26,
  ITCN_B_TIM2_TOF_SOURCE_27,
  ITCN_B_TIM2_PAIF_SOURCE_28,
  ITCN_B_TIM2_PAOVF_SOURCE_29,
  ITCN_B_PIT1_PIF_SOURCE_30,
  ITCN_B_PIT2_PIF_SOURCE_31,
  ITCN_B_EDGEPORT_EPF0_SOURCE_32,
  ITCN_B_EDGEPORT_EPF1_SOURCE_33,
  ITCN_B_EDGEPORT_EPF2_SOURCE_34,
  ITCN_B_EDGEPORT_EPF3_SOURCE_35,
  ITCN_B_EDGEPORT_EPF4_SOURCE_36,
  ITCN_B_EDGEPORT_EPF5_SOURCE_37,
  ITCN_B_EDGEPORT_EPF6_SOURCE_38,
  ITCN_B_EDGEPORT_EPF7_SOURCE_39
} ITCN_B_InterruptSource_t;
```

9.3.4.9 ITCN_B_InterruptControl_t – Enable or disable interrupts

```
typedef enum
{
  ITCN_B_INTERRUPT_DISABLE,
  ITCN_B_INTERRUPT_ENABLE
} ITCN_B_InterruptControl_t;
```

9.3.4.10 ITCN_B_Force_t – Force or clear interrupts

```
typedef enum
```

```
{
  ITCN_B_CLEAR_INTERRUPT,
  ITCN_B_FORCE_INTERRUPT,
} ITCN_B_Force_t;
```

9.3.5 Register Macros

9.3.5.1 Interrupt Control Register (ICR) Bits & Masks

```
#define ICR_AE_BITNO   15           /* autovector enable bit    */
#define ICR_AE_MASK    (1 << ICR_AE_BITNO)
#define ICR_FVE_BITNO  14           /* fast vector enable bit    */
#define ICR_FVE_MASK   (1 << ICR_FVE_BITNO)
#define ICR_ME_BITNO   13          /* mask enable bit    */
#define ICR_ME_MASK    (1 << ICR_ME_BITNO)
#define ICR_MFI_BITNO  12          /* mask fast interrupts bit */
#define ICR_MFI_MASK   (1 << ICR_MFI_BITNO)
#define ICR_MASK_BITNO 0           /* interrupt mask value */
#define ICR_MASK_MAX   0x1F
#define ICR_MASK_MASK  (ICR_MASK_MAX << ICR_MASK_BITNO)

#define ICR_RESET_MASK (0x8000)
```

9.3.5.2 Interrupt Status Register (ISR) Bits & Masks

```
#define ISR_INT_BITNO  9           /* normal interrupt request */
#define ISR_INT_MASK   (1 << ISR_INT_BITNO)
#define ISR_FINT_BITNO 8           /* fast interrupt request */
#define ISR_FINT_MASK  (1 << ISR_FINT_BITNO)
#define ISR_VEC_BITNO  0           /* interrupt vector number */
#define ISR_VEC_MAX    0x3F
#define ISR_VEC_MASK   (ISR_VEC_MAX << ISR_VEC_BITNO)

#define ISR_RESET_MASK 0x0000
```

9.3.5.3 Interrupt Force Register High (IFRH) Bits & Masks

```
#define IFRH_IF39_BITNO 7
#define IFRH_IF39_MASK  (1 << IFRH_IF39_BITNO)
#define IFRH_IF38_BITNO 6
#define IFRH_IF38_MASK  (1 << IFRH_IF38_BITNO)
#define IFRH_IF37_BITNO 5
#define IFRH_IF37_MASK  (1 << IFRH_IF37_BITNO)
#define IFRH_IF36_BITNO 4
#define IFRH_IF36_MASK  (1 << IFRH_IF36_BITNO)
#define IFRH_IF35_BITNO 3
#define IFRH_IF35_MASK  (1 << IFRH_IF35_BITNO)
#define IFRH_IF34_BITNO 2
#define IFRH_IF34_MASK  (1 << IFRH_IF34_BITNO)
#define IFRH_IF33_BITNO 1
#define IFRH_IF33_MASK  (1 << IFRH_IF33_BITNO)
#define IFRH_IF32_BITNO 0
```

```
#define IFRH_IF32_MASK  (1 << IFRH_IF32_BITNO)

#define IFRH_IF_MASK    (0xFF << IFRH_IF32_BITNO)
#define IFRH_RESET_MASK (0x00000000)
```

9.3.5.4 Interrupt Force Register Low (IFRL) Bits & Masks

```
#define IFRL_IF31_BITNO 31
#define IFRL_IF31_MASK  (1 << IFRL_IF31_BITNO)
#define IFRL_IF30_BITNO 30
#define IFRL_IF30_MASK  (1 << IFRL_IF30_BITNO)
#define IFRL_IF29_BITNO 29
#define IFRL_IF29_MASK  (1 << IFRL_IF29_BITNO)
#define IFRL_IF28_BITNO 28
#define IFRL_IF28_MASK  (1 << IFRL_IF28_BITNO)
#define IFRL_IF27_BITNO 27
#define IFRL_IF27_MASK  (1 << IFRL_IF27_BITNO)
#define IFRL_IF26_BITNO 26
#define IFRL_IF26_MASK  (1 << IFRL_IF26_BITNO)
#define IFRL_IF25_BITNO 25
#define IFRL_IF25_MASK  (1 << IFRL_IF25_BITNO)
#define IFRL_IF24_BITNO 24
#define IFRL_IF24_MASK  (1 << IFRL_IF24_BITNO)
#define IFRL_IF23_BITNO 23
#define IFRL_IF23_MASK  (1 << IFRL_IF23_BITNO)
#define IFRL_IF22_BITNO 22
#define IFRL_IF22_MASK  (1 << IFRL_IF22_BITNO)
#define IFRL_IF21_BITNO 21
#define IFRL_IF21_MASK  (1 << IFRL_IF21_BITNO)
#define IFRL_IF20_BITNO 20
#define IFRL_IF20_MASK  (1 << IFRL_IF20_BITNO)
#define IFRL_IF19_BITNO 19
#define IFRL_IF19_MASK  (1 << IFRL_IF19_BITNO)
#define IFRL_IF18_BITNO 18
#define IFRL_IF18_MASK  (1 << IFRL_IF18_BITNO)
#define IFRL_IF17_BITNO 17
#define IFRL_IF17_MASK  (1 << IFRL_IF17_BITNO)
#define IFRL_IF16_BITNO 16
#define IFRL_IF16_MASK  (1 << IFRL_IF16_BITNO)
#define IFRL_IF15_BITNO 15
#define IFRL_IF15_MASK  (1 << IFRL_IF15_BITNO)
#define IFRL_IF14_BITNO 14
#define IFRL_IF14_MASK  (1 << IFRL_IF14_BITNO)
#define IFRL_IF13_BITNO 13
#define IFRL_IF13_MASK  (1 << IFRL_IF13_BITNO)
#define IFRL_IF12_BITNO 12
#define IFRL_IF12_MASK  (1 << IFRL_IF12_BITNO)
#define IFRL_IF11_BITNO 11
#define IFRL_IF11_MASK  (1 << IFRL_IF11_BITNO)
#define IFRL_IF10_BITNO 10
#define IFRL_IF10_MASK  (1 << IFRL_IF10_BITNO)
#define IFRL_IF9_BITNO  9
#define IFRL_IF9_MASK   (1 << IFRL_IF9_BITNO)
```

```
#define IFRL_IF8_BITNO  8
#define IFRL_IF8_MASK   (1 << IFRL_IF8_BITNO)
#define IFRL_IF7_BITNO  7
#define IFRL_IF7_MASK   (1 << IFRL_IF7_BITNO)
#define IFRL_IF6_BITNO  6
#define IFRL_IF6_MASK   (1 << IFRL_IF6_BITNO)
#define IFRL_IF5_BITNO  5
#define IFRL_IF5_MASK   (1 << IFRL_IF5_BITNO)
#define IFRL_IF4_BITNO  4
#define IFRL_IF4_MASK   (1 << IFRL_IF4_BITNO)
#define IFRL_IF3_BITNO  3
#define IFRL_IF3_MASK   (1 << IFRL_IF3_BITNO)
#define IFRL_IF2_BITNO  2
#define IFRL_IF2_MASK   (1 << IFRL_IF2_BITNO)
#define IFRL_IF1_BITNO  1
#define IFRL_IF1_MASK   (1 << IFRL_IF1_BITNO)
#define IFRL_IF0_BITNO  0
#define IFRL_IF0_MASK   (1 << IFRL_IF0_BITNO)

#define IFRL_IF_MASK    (0xFFFFFFFF)
#define IFRL_RESET_MASK (0x00000000)
```

9.3.5.5 Interrupt Pending Register (IPR) Bits & Masks

```
#define IPR_IP31_BITNO 31
#define IPR_IP31_MASK  (1 << IPR_IP31_BITNO)
#define IPR_IP30_BITNO 30
#define IPR_IP30_MASK  (1 << IPR_IP30_BITNO)
#define IPR_IP29_BITNO 29
#define IPR_IP29_MASK  (1 << IPR_IP29_BITNO)
#define IPR_IP28_BITNO 28
#define IPR_IP28_MASK  (1 << IPR_IP28_BITNO)
#define IPR_IP27_BITNO 27
#define IPR_IP27_MASK  (1 << IPR_IP27_BITNO)
#define IPR_IP26_BITNO 26
#define IPR_IP26_MASK  (1 << IPR_IP26_BITNO)
#define IPR_IP25_BITNO 25
#define IPR_IP25_MASK  (1 << IPR_IP25_BITNO)
#define IPR_IP24_BITNO 24
#define IPR_IP24_MASK  (1 << IPR_IP24_BITNO)
#define IPR_IP23_BITNO 23
#define IPR_IP23_MASK  (1 << IPR_IP23_BITNO)
#define IPR_IP22_BITNO 22
#define IPR_IP22_MASK  (1 << IPR_IP22_BITNO)
#define IPR_IP21_BITNO 21
#define IPR_IP21_MASK  (1 << IPR_IP21_BITNO)
#define IPR_IP20_BITNO 20
#define IPR_IP20_MASK  (1 << IPR_IP20_BITNO)
#define IPR_IP19_BITNO 19
#define IPR_IP19_MASK  (1 << IPR_IP19_BITNO)
#define IPR_IP18_BITNO 18
#define IPR_IP18_MASK  (1 << IPR_IP18_BITNO)
#define IPR_IP17_BITNO 17
```

```
#define IPR_IP17_MASK  (1 << IPR_IP17_BITNO)
#define IPR_IP16_BITNO 16
#define IPR_IP16_MASK  (1 << IPR_IP16_BITNO)
#define IPR_IP15_BITNO 15
#define IPR_IP15_MASK  (1 << IPR_IP15_BITNO)
#define IPR_IP14_BITNO 14
#define IPR_IP14_MASK  (1 << IPR_IP14_BITNO)
#define IPR_IP13_BITNO 13
#define IPR_IP13_MASK  (1 << IPR_IP13_BITNO)
#define IPR_IP12_BITNO 12
#define IPR_IP12_MASK  (1 << IPR_IP12_BITNO)
#define IPR_IP11_BITNO 11
#define IPR_IP11_MASK  (1 << IPR_IP11_BITNO)
#define IPR_IP10_BITNO 10
#define IPR_IP10_MASK  (1 << IPR_IP10_BITNO)
#define IPR_IP9_BITNO  9
#define IPR_IP9_MASK   (1 << IPR_IP9_BITNO)
#define IPR_IP8_BITNO  8
#define IPR_IP8_MASK   (1 << IPR_IP8_BITNO)
#define IPR_IP7_BITNO  7
#define IPR_IP7_MASK   (1 << IPR_IP7_BITNO)
#define IPR_IP6_BITNO  6
#define IPR_IP6_MASK   (1 << IPR_IP6_BITNO)
#define IPR_IP5_BITNO  5
#define IPR_IP5_MASK   (1 << IPR_IP5_BITNO)
#define IPR_IP4_BITNO  4
#define IPR_IP4_MASK   (1 << IPR_IP4_BITNO)
#define IPR_IP3_BITNO  3
#define IPR_IP3_MASK   (1 << IPR_IP3_BITNO)
#define IPR_IP2_BITNO  2
#define IPR_IP2_MASK   (1 << IPR_IP2_BITNO)
#define IPR_IP1_BITNO  1
#define IPR_IP1_MASK   (1 << IPR_IP1_BITNO)
#define IPR_IP0_BITNO  0
#define IPR_IP0_MASK   (1 << IPR_IP0_BITNO)

#define IPR_IP_MASK    (0xFFFFFFFF)
#define IPR_RESET_MASK (0x00000000)
```

9.3.5.6 Normal Interrupt Enable Register (NIER) Bits & Masks

```
#define NIER_NIE31_BITNO 31
#define NIER_NIE31_MASK  (1 << NIER_NIE31_BITNO)
#define NIER_NIE30_BITNO 30
#define NIER_NIE30_MASK  (1 << NIER_NIE30_BITNO)
#define NIER_NIE29_BITNO 29
#define NIER_NIE29_MASK  (1 << NIER_NIE29_BITNO)
#define NIER_NIE28_BITNO 28
#define NIER_NIE28_MASK  (1 << NIER_NIE28_BITNO)
#define NIER_NIE27_BITNO 27
#define NIER_NIE27_MASK  (1 << NIER_NIE27_BITNO)
#define NIER_NIE26_BITNO 26
#define NIER_NIE26_MASK  (1 << NIER_NIE26_BITNO)
```

```
#define NIER_NIE25_BITNO 25
#define NIER_NIE25_MASK  (1 << NIER_NIE25_BITNO)
#define NIER_NIE24_BITNO 24
#define NIER_NIE24_MASK  (1 << NIER_NIE24_BITNO)
#define NIER_NIE23_BITNO 23
#define NIER_NIE23_MASK  (1 << NIER_NIE23_BITNO)
#define NIER_NIE22_BITNO 22
#define NIER_NIE22_MASK  (1 << NIER_NIE22_BITNO)
#define NIER_NIE21_BITNO 21
#define NIER_NIE21_MASK  (1 << NIER_NIE21_BITNO)
#define NIER_NIE20_BITNO 20
#define NIER_NIE20_MASK  (1 << NIER_NIE20_BITNO)
#define NIER_NIE19_BITNO 19
#define NIER_NIE19_MASK  (1 << NIER_NIE19_BITNO)
#define NIER_NIE18_BITNO 18
#define NIER_NIE18_MASK  (1 << NIER_NIE18_BITNO)
#define NIER_NIE17_BITNO 17
#define NIER_NIE17_MASK  (1 << NIER_NIE17_BITNO)
#define NIER_NIE16_BITNO 16
#define NIER_NIE16_MASK  (1 << NIER_NIE16_BITNO)
#define NIER_NIE15_BITNO 15
#define NIER_NIE15_MASK  (1 << NIER_NIE15_BITNO)
#define NIER_NIE14_BITNO 14
#define NIER_NIE14_MASK  (1 << NIER_NIE14_BITNO)
#define NIER_NIE13_BITNO 13
#define NIER_NIE13_MASK  (1 << NIER_NIE13_BITNO)
#define NIER_NIE12_BITNO 12
#define NIER_NIE12_MASK  (1 << NIER_NIE12_BITNO)
#define NIER_NIE11_BITNO 11
#define NIER_NIE11_MASK  (1 << NIER_NIE11_BITNO)
#define NIER_NIE10_BITNO 10
#define NIER_NIE10_MASK  (1 << NIER_NIE10_BITNO)
#define NIER_NIE9_BITNO  9
#define NIER_NIE9_MASK   (1 << NIER_NIE9_BITNO)
#define NIER_NIE8_BITNO  8
#define NIER_NIE8_MASK   (1 << NIER_NIE8_BITNO)
#define NIER_NIE7_BITNO  7
#define NIER_NIE7_MASK   (1 << NIER_NIE7_BITNO)
#define NIER_NIE6_BITNO  6
#define NIER_NIE6_MASK   (1 << NIER_NIE6_BITNO)
#define NIER_NIE5_BITNO  5
#define NIER_NIE5_MASK   (1 << NIER_NIE5_BITNO)
#define NIER_NIE4_BITNO  4
#define NIER_NIE4_MASK   (1 << NIER_NIE4_BITNO)
#define NIER_NIE3_BITNO  3
#define NIER_NIE3_MASK   (1 << NIER_NIE3_BITNO)
#define NIER_NIE2_BITNO  2
#define NIER_NIE2_MASK   (1 << NIER_NIE2_BITNO)
#define NIER_NIE1_BITNO  1
#define NIER_NIE1_MASK   (1 << NIER_NIE1_BITNO)
#define NIER_NIE0_BITNO  0
#define NIER_NIE0_MASK   (1 << NIER_NIE0_BITNO)

#define NIER_NIE_MASK    (0xFFFFFFFF)
```

```
#define NIER_RESET_MASK  (0x00000000)
```

9.3.5.7 Normal Interrupt Pending Register (NIPR) Bits & Masks

```
#define NIPR_NIP31_BITNO 31
#define NIPR_NIP31_MASK  (1 << NIPR_NIP31_BITNO)
#define NIPR_NIP30_BITNO 30
#define NIPR_NIP30_MASK  (1 << NIPR_NIP30_BITNO)
#define NIPR_NIP29_BITNO 29
#define NIPR_NIP29_MASK  (1 << NIPR_NIP29_BITNO)
#define NIPR_NIP28_BITNO 28
#define NIPR_NIP28_MASK  (1 << NIPR_NIP28_BITNO)
#define NIPR_NIP27_BITNO 27
#define NIPR_NIP27_MASK  (1 << NIPR_NIP27_BITNO)
#define NIPR_NIP26_BITNO 26
#define NIPR_NIP26_MASK  (1 << NIPR_NIP26_BITNO)
#define NIPR_NIP25_BITNO 25
#define NIPR_NIP25_MASK  (1 << NIPR_NIP25_BITNO)
#define NIPR_NIP24_BITNO 24
#define NIPR_NIP24_MASK  (1 << NIPR_NIP24_BITNO)
#define NIPR_NIP23_BITNO 23
#define NIPR_NIP23_MASK  (1 << NIPR_NIP23_BITNO)
#define NIPR_NIP22_BITNO 22
#define NIPR_NIP22_MASK  (1 << NIPR_NIP22_BITNO)
#define NIPR_NIP21_BITNO 21
#define NIPR_NIP21_MASK  (1 << NIPR_NIP21_BITNO)
#define NIPR_NIP20_BITNO 20
#define NIPR_NIP20_MASK  (1 << NIPR_NIP20_BITNO)
#define NIPR_NIP19_BITNO 19
#define NIPR_NIP19_MASK  (1 << NIPR_NIP19_BITNO)
#define NIPR_NIP18_BITNO 18
#define NIPR_NIP18_MASK  (1 << NIPR_NIP18_BITNO)
#define NIPR_NIP17_BITNO 17
#define NIPR_NIP17_MASK  (1 << NIPR_NIP17_BITNO)
#define NIPR_NIP16_BITNO 16
#define NIPR_NIP16_MASK  (1 << NIPR_NIP16_BITNO)
#define NIPR_NIP15_BITNO 15
#define NIPR_NIP15_MASK  (1 << NIPR_NIP15_BITNO)
#define NIPR_NIP14_BITNO 14
#define NIPR_NIP14_MASK  (1 << NIPR_NIP14_BITNO)
#define NIPR_NIP13_BITNO 13
#define NIPR_NIP13_MASK  (1 << NIPR_NIP13_BITNO)
#define NIPR_NIP12_BITNO 12
#define NIPR_NIP12_MASK  (1 << NIPR_NIP12_BITNO)
#define NIPR_NIP11_BITNO 11
#define NIPR_NIP11_MASK  (1 << NIPR_NIP11_BITNO)
#define NIPR_NIP10_BITNO 10
#define NIPR_NIP10_MASK  (1 << NIPR_NIP10_BITNO)
#define NIPR_NIP9_BITNO  9
#define NIPR_NIP9_MASK   (1 << NIPR_NIP9_BITNO)
#define NIPR_NIP8_BITNO  8
#define NIPR_NIP8_MASK   (1 << NIPR_NIP8_BITNO)
#define NIPR_NIP7_BITNO  7
```

```
#define NIPR_NIP7_MASK   (1 << NIPR_NIP7_BITNO)
#define NIPR_NIP6_BITNO  6
#define NIPR_NIP6_MASK   (1 << NIPR_NIP6_BITNO)
#define NIPR_NIP5_BITNO  5
#define NIPR_NIP5_MASK   (1 << NIPR_NIP5_BITNO)
#define NIPR_NIP4_BITNO  4
#define NIPR_NIP4_MASK   (1 << NIPR_NIP4_BITNO)
#define NIPR_NIP3_BITNO  3
#define NIPR_NIP3_MASK   (1 << NIPR_NIP3_BITNO)
#define NIPR_NIP2_BITNO  2
#define NIPR_NIP2_MASK   (1 << NIPR_NIP2_BITNO)
#define NIPR_NIP1_BITNO  1
#define NIPR_NIP1_MASK   (1 << NIPR_NIP1_BITNO)
#define NIPR_NIP0_BITNO  0
#define NIPR_NIP0_MASK   (1 << NIPR_NIP0_BITNO)

#define NIPR_NIP_MASK    (0xFFFFFFFF)
#define NIPR_RESET_MASK  (0x00000000)
```

9.3.5.8 Fast Interrupt Enable Register (FIER) Bits & Masks

```
#define FIER_FIE31_BITNO 31
#define FIER_FIE31_MASK  (1 << FIER_FIE31_BITNO)
#define FIER_FIE30_BITNO 30
#define FIER_FIE30_MASK  (1 << FIER_FIE30_BITNO)
#define FIER_FIE29_BITNO 29
#define FIER_FIE29_MASK  (1 << FIER_FIE29_BITNO)
#define FIER_FIE28_BITNO 28
#define FIER_FIE28_MASK  (1 << FIER_FIE28_BITNO)
#define FIER_FIE27_BITNO 27
#define FIER_FIE27_MASK  (1 << FIER_FIE27_BITNO)
#define FIER_FIE26_BITNO 26
#define FIER_FIE26_MASK  (1 << FIER_FIE26_BITNO)
#define FIER_FIE25_BITNO 25
#define FIER_FIE25_MASK  (1 << FIER_FIE25_BITNO)
#define FIER_FIE24_BITNO 24
#define FIER_FIE24_MASK  (1 << FIER_FIE24_BITNO)
#define FIER_FIE23_BITNO 23
#define FIER_FIE23_MASK  (1 << FIER_FIE23_BITNO)
#define FIER_FIE22_BITNO 22
#define FIER_FIE22_MASK  (1 << FIER_FIE22_BITNO)
#define FIER_FIE21_BITNO 21
#define FIER_FIE21_MASK  (1 << FIER_FIE21_BITNO)
#define FIER_FIE20_BITNO 20
#define FIER_FIE20_MASK  (1 << FIER_FIE20_BITNO)
#define FIER_FIE19_BITNO 19
#define FIER_FIE19_MASK  (1 << FIER_FIE19_BITNO)
#define FIER_FIE18_BITNO 18
#define FIER_FIE18_MASK  (1 << FIER_FIE18_BITNO)
#define FIER_FIE17_BITNO 17
#define FIER_FIE17_MASK  (1 << FIER_FIE17_BITNO)
#define FIER_FIE16_BITNO 16
#define FIER_FIE16_MASK  (1 << FIER_FIE16_BITNO)
```

```
#define FIER_FIE15_BITNO 15
#define FIER_FIE15_MASK  (1 << FIER_FIE15_BITNO)
#define FIER_FIE14_BITNO 14
#define FIER_FIE14_MASK  (1 << FIER_FIE14_BITNO)
#define FIER_FIE13_BITNO 13
#define FIER_FIE13_MASK  (1 << FIER_FIE13_BITNO)
#define FIER_FIE12_BITNO 12
#define FIER_FIE12_MASK  (1 << FIER_FIE12_BITNO)
#define FIER_FIE11_BITNO 11
#define FIER_FIE11_MASK  (1 << FIER_FIE11_BITNO)
#define FIER_FIE10_BITNO 10
#define FIER_FIE10_MASK  (1 << FIER_FIE10_BITNO)
#define FIER_FIE9_BITNO  9
#define FIER_FIE9_MASK   (1 << FIER_FIE9_BITNO)
#define FIER_FIE8_BITNO  8
#define FIER_FIE8_MASK   (1 << FIER_FIE8_BITNO)
#define FIER_FIE7_BITNO  7
#define FIER_FIE7_MASK   (1 << FIER_FIE7_BITNO)
#define FIER_FIE6_BITNO  6
#define FIER_FIE6_MASK   (1 << FIER_FIE6_BITNO)
#define FIER_FIE5_BITNO  5
#define FIER_FIE5_MASK   (1 << FIER_FIE5_BITNO)
#define FIER_FIE4_BITNO  4
#define FIER_FIE4_MASK   (1 << FIER_FIE4_BITNO)
#define FIER_FIE3_BITNO  3
#define FIER_FIE3_MASK   (1 << FIER_FIE3_BITNO)
#define FIER_FIE2_BITNO  2
#define FIER_FIE2_MASK   (1 << FIER_FIE2_BITNO)
#define FIER_FIE1_BITNO  1
#define FIER_FIE1_MASK   (1 << FIER_FIE1_BITNO)
#define FIER_FIE0_BITNO  0
#define FIER_FIE0_MASK   (1 << FIER_FIE0_BITNO)

#define FIER_FIE_MASK    (0xFFFFFFFF)
#define FIER_RESET_MASK  (0x00000000)
```

9.3.5.9 Fast Interrupt Pending Register (FIPR) Bits & Masks

```
#define FIPR_FIP31_BITNO 31
#define FIPR_FIP31_MASK  (1 << FIPR_FIP31_BITNO)
#define FIPR_FIP30_BITNO 30
#define FIPR_FIP30_MASK  (1 << FIPR_FIP30_BITNO)
#define FIPR_FIP29_BITNO 29
#define FIPR_FIP29_MASK  (1 << FIPR_FIP29_BITNO)
#define FIPR_FIP28_BITNO 28
#define FIPR_FIP28_MASK  (1 << FIPR_FIP28_BITNO)
#define FIPR_FIP27_BITNO 27
#define FIPR_FIP27_MASK  (1 << FIPR_FIP27_BITNO)
#define FIPR_FIP26_BITNO 26
#define FIPR_FIP26_MASK  (1 << FIPR_FIP26_BITNO)
#define FIPR_FIP25_BITNO 25
#define FIPR_FIP25_MASK  (1 << FIPR_FIP25_BITNO)
#define FIPR_FIP24_BITNO 24
```

```
#define FIPR_FIP24_MASK  (1 << FIPR_FIP24_BITNO)
#define FIPR_FIP23_BITNO 23
#define FIPR_FIP23_MASK  (1 << FIPR_FIP23_BITNO)
#define FIPR_FIP22_BITNO 22
#define FIPR_FIP22_MASK  (1 << FIPR_FIP22_BITNO)
#define FIPR_FIP21_BITNO 21
#define FIPR_FIP21_MASK  (1 << FIPR_FIP21_BITNO)
#define FIPR_FIP20_BITNO 20
#define FIPR_FIP20_MASK  (1 << FIPR_FIP20_BITNO)
#define FIPR_FIP19_BITNO 19
#define FIPR_FIP19_MASK  (1 << FIPR_FIP19_BITNO)
#define FIPR_FIP18_BITNO 18
#define FIPR_FIP18_MASK  (1 << FIPR_FIP18_BITNO)
#define FIPR_FIP17_BITNO 17
#define FIPR_FIP17_MASK  (1 << FIPR_FIP17_BITNO)
#define FIPR_FIP16_BITNO 16
#define FIPR_FIP16_MASK  (1 << FIPR_FIP16_BITNO)
#define FIPR_FIP15_BITNO 15
#define FIPR_FIP15_MASK  (1 << FIPR_FIP15_BITNO)
#define FIPR_FIP14_BITNO 14
#define FIPR_FIP14_MASK  (1 << FIPR_FIP14_BITNO)
#define FIPR_FIP13_BITNO 13
#define FIPR_FIP13_MASK  (1 << FIPR_FIP13_BITNO)
#define FIPR_FIP12_BITNO 12
#define FIPR_FIP12_MASK  (1 << FIPR_FIP12_BITNO)
#define FIPR_FIP11_BITNO 11
#define FIPR_FIP11_MASK  (1 << FIPR_FIP11_BITNO)
#define FIPR_FIP10_BITNO 10
#define FIPR_FIP10_MASK  (1 << FIPR_FIP10_BITNO)
#define FIPR_FIP9_BITNO  9
#define FIPR_FIP9_MASK   (1 << FIPR_FIP9_BITNO)
#define FIPR_FIP8_BITNO  8
#define FIPR_FIP8_MASK   (1 << FIPR_FIP8_BITNO)
#define FIPR_FIP7_BITNO  7
#define FIPR_FIP7_MASK   (1 << FIPR_FIP7_BITNO)
#define FIPR_FIP6_BITNO  6
#define FIPR_FIP6_MASK   (1 << FIPR_FIP6_BITNO)
#define FIPR_FIP5_BITNO  5
#define FIPR_FIP5_MASK   (1 << FIPR_FIP5_BITNO)
#define FIPR_FIP4_BITNO  4
#define FIPR_FIP4_MASK   (1 << FIPR_FIP4_BITNO)
#define FIPR_FIP3_BITNO  3
#define FIPR_FIP3_MASK   (1 << FIPR_FIP3_BITNO)
#define FIPR_FIP2_BITNO  2
#define FIPR_FIP2_MASK   (1 << FIPR_FIP2_BITNO)
#define FIPR_FIP1_BITNO  1
#define FIPR_FIP1_MASK   (1 << FIPR_FIP1_BITNO)
#define FIPR_FIP0_BITNO  0
#define FIPR_FIP0_MASK   (1 << FIPR_FIP0_BITNO)

#define FIPR_FIP_MASK    (0xFFFFFFFF)
#define FIPR_RESET_MASK  (0x00000000)
```

9.3.5.10 Priority Level Select Register (PLSR) Bits & Masks

```
#define PLSR_PLS_BITNO  0
#define PLSR_PLS_MAX    0x1F
#define PLSR_PLS_MASK   (PLSR0_PLS_MAX << PLSR0_PLS_BITNO)

#define PLSR_RESET_MASK (0x00)
```

9.3.6 Miscellaneous Macros

```
/* vector table positions */
#define AUTOVECTOR_NORMAL_INTERRUPTS 10
#define AUTOVECTOR_FAST_INTERRUPTS 11
#define VECTOR_NORMAL_INTERRUPTS_BEGIN 32
#define VECTOR_NORMAL_INTERRUPTS_END 63
#define VECTOR_FAST_INTERRUPTS_BEGIN 64
#define VECTOR_FAST_INTERRUPTS_END 95

/* priority level macros */
#define INTERRUPT_PRIORITY_LEVEL_MIN 0
#define INTERRUPT_PRIORITY_LEVEL_MAX 31

/* register misc. macros */
#define REGISTER_8BIT_MAX 0xFF
#define REGISTER_16BIT_MAX 0xFFFF
#define REGISTER_32BIT_MAX 0xFFFFFFFF

#define REGISTER_8BIT_MIN 0
#define REGISTER_16BIT_MIN 0
#define REGISTER_32BIT_MIN 0

#define REGISTER_8BIT_BITS 8
#define REGISTER_16BIT_BITS 16
#define REGISTER_32BIT_BITS 32
```

9.3.7 Return Codes

Return codes from the enumerated type “ITCN_B_ReturnCode_t”.

Return Code	Functions Returning	Description
ITCN_B_ERR_NONE	All	No error.
ITCN_B_ERR_INVALID_HANDLE	All	Invalid device handle.
ITCN_B_ERR_INVALID_VECTOR_TYPE	Init, SetInterruptServiceRoutine	Invalid vector type (autovectored or vectored interrupt).
ITCN_B_ERR_INVALID_FAST_INTERRUPT_CONTROL	Init	Invalid fast interrupt control selection.
ITCN_B_ERR_INVALID_MASK_MODE	Init	Invalid mask mode.
ITCN_B_ERR_INVALID_MASK_LEVEL	Init	Invalid mask level.
ITCN_B_ERR_INVALID_PRIORITY_LEVEL	SetInterruptServiceRoutine, ControlOperation	Invalid priority level.
ITCN_B_ERR_INVALID_INTERRUPT_TYPE	SetInterruptServiceRoutine, ControlOperation	Invalid interrupt type (fast or normal).
ITCN_B_ERR_INVALID_ISR_ADDRESS	SetInterruptServiceRoutine	Address to ISR is zero.
ITCN_B_ERR_INVALID_INTERRUPT_SOURCE	SetInterruptServiceRoutine, ForceInterrupt	Interrupt source is not valid.
ITCN_B_ERR_INVALID_INTERRUPT_CONTROL	ControlOperation	Interrupt control (enable or disable) is not valid.
ITCN_B_ERR_INVALID_FORCE_CONTROL	ForceInterrupt	Force control selection is invalid
ITCN_B_ERR_BAD_RESULT_ADDR	GetStatus, GetRegister	Result Pointer is zero
ITCN_B_ERR_INVALID_REGISTER	SetRegister, GetRegister	ITCN register selection is invalid
ITCN_B_ERR_INVALID_REGISTER_VALUE	SetRegister	Register value is out of range for register selected

9.4 ITCN_B API Definitions

9.4.1 ITCN_B_Init

9.4.1.1 Description

The ITCN_B_Init function initializes the interrupt controller to a user-determined condition. It enables vectored or autovectored interrupts, enables or disables fast interrupts, and specifies interrupts masking. Parameter checking is a compile-time option.

9.4.1.2 Prototype

```
ITCN_B_ReturnCode_t  ITCN_B_Init (
          pITCN_B_t                          ITCNPtr,
          ITCN_B_Autovector_t                vectorType,
          ITCN_B_FastInterruptControl_t      fastInterruptControl,
          ITCN_B_Mask_t                      maskMode,
          UINT8                              maskLevel
)
```

9.4.1.3 Arguments

Data Type	Formal Name	Description	Valid Values
pITCN_B_t	ITCNPtr	ITCN base address associated with this driver.	Any non-zero core processor data address.
ITCN_B_Autovector_t	vectorType	specify vectored interrupt enable or autovectored interrupt enable	ITCN_B_VECTORED_INTERRUPTS, ITCN_B_AUTOVECTORED_INTERRUPTS
ITCN_B_FastInterruptControl_t	fastInterruptControl	enable or disable fast interrupts	ITCN_B_FAST_INTERRUPT_DISABLE, ITCN_B_FAST_INTERRUPT_ENABLE
ITCN_B_Mask_t	maskMode	specify mask mode.	ITCN_B_MASK_DISABLE, ITCN_B_MASK_ENABLE_ONLY_NORMAL, ITCN_B_MASK_ENABLE_FAST_AND_NORMAL
UINT8	maskLevel	specify the highest interrupt priority level to be masked	0, 1, 2, …, 31

9.4.1.4 Return Values

Error Code	Description
ITCN_B_ERR_NONE	No error
ITCN_B_ERR_INVALID_HANDLE	ITCN base address parameter is zero.
ITCN_B_ERR_INVALID_VECTOR_TYPE	Invalid vector type (autovectored or vectored interrupt).
ITCN_B_ERR_INVALID_FAST_INTERRUPT_CONTROL	Invalid fast interrupt control selection.
ITCN_B_ERR_INVALID_MASK_MODE	Invalid mask mode.
ITCN_B_ERR_INVALID_MASK_LEVEL	Invalid mask level.

9.4.2 ITCN_B_InitVectorTable

9.4.2.1 Description

The ITCN_B_InitVectorTable function initalizes the vector table with user-supplied vector values for fast and normal interrupts. Parameter checking is a compile-time option.

9.4.2.2 Prototype

```
ITCN_B_ReturnCode_t ITCN_B_InitVectorTable (
          pITCN_B_t     ITCNPtr,
          UINT32        normISR,
          UINT32        fastISR
)
```

9.4.2.3 Arguments

Data Type	Formal Name	Description	Valid Values
pITCN_B_t	ITCNPtr	ITCN base address associated with this driver.	Any non-zero core processor data address.
UINT32	normISR	Pointer to user-specified normal Interrupt Service Routine	0x00000000 – 0xFFFFFFFF
UINT32	fastISR	Pointer to user-specified fast Interrupt Service Routine	0x00000000 – 0xFFFFFFFF

9.4.2.4 Return Values

Error Code	Description
ITCN_B_ERR_NONE	No error
ITCN_B_ERR_INVALID_HANDLE	ITCN base address parameter is zero

9.4.3 ITCN_B_SetInterruptServiceRoutine

9.4.3.1 Description

The ITCN_B_SetInterruptServiceRoutine function sets the priority level for a selected interrupt source and sets a custom ISR value in the vector table for this interrupt source. Parameter checking is a compile-time option.

9.4.3.2 Prototype

```
ITCN_B_ReturnCode_t ITCN_B_SetInterruptServiceRoutine (
          pITCN_B_t                 ITCNPtr,
          UINT8                     priorityLevel,
          ITCN_B_InterruptType_t    interruptType,
          UINT32                    ISRAddress,
          ITCN_B_InterruptSource_t  interruptSource,
          ITCN_B_Autovector_t       vectorType
)
```

9.4.3.3 Arguments

Data Type	Formal Name	Description	Valid Values
pITCN_B_t	ITCNPtr	ITCN base address associated with this driver.	Any non-zero core processor data address.
UINT8	priorityLevel	Priority level for this interrupt.	0, 1, …, 31
ITCN_B_InterruptType_t	interruptType	Specify fast or normal interrupt.	ITCN_B_INTERRUPT_NORMAL, ITCN_B_INTERRUPT_FAST
UINT32	ISRAddress	Pointer to user-specified Interrupt Service Routine.	0x00000000 – 0xFFFFFFFF
ITCN_B_InterruptSource_t	interruptSource	Source of interrupt.	ITCN_B_ADC_PF1_SOURCE_0, ITCN_B_ADC_CF1_SOURCE_1, ITCN_B_ADC_PF2_SOURCE_2, ITCN_B_ADC_CF2_SOURCE_3, ITCN_B_SPI_MODF_SOURCE_4, ITCN_B_SPI_SPIF_SOURCE_5, ITCN_B_SCI1_TDRE_SOURCE_6, ITCN_B_SCI1_TC_SOURCE_7, ITCN_B_SCI1_RDRF_SOURCE_8, ITCN_B_SCI1_OR_SOURCE_9, ITCN_B_SCI1_IDLE_SOURCE_10, ITCN_B_SCI2_TDRE_SOURCE_11, ITCN_B_SCI2_TC_SOURCE_12, ITCN_B_SCI2_RDRF_SOURCE_13, ITCN_B_SCI2_OR_SOURCE_14, ITCN_B_SCI2_IDLE_SOURCE_15, ITCN_B_TIM1_C0F_SOURCE_16, ITCN_B_TIM1_C1F_SOURCE_17, ITCN_B_TIM1_C2F_SOURCE_18, ITCN_B_TIM1_C3F_SOURCE_19, ITCN_B_TIM1_TOF_SOURCE_20, ITCN_B_TIM1_PAIF_SOURCE_21, ITCN_B_TIM1_PAOVF_SOURCE_22, ITCN_B_TIM2_C0F_SOURCE_23, ITCN_B_TIM2_C1F_SOURCE_24, ITCN_B_TIM2_C2F_SOURCE_25, ITCN_B_TIM2_C3F_SOURCE_26, ITCN_B_TIM2_TOF_SOURCE_27, ITCN_B_TIM2_PAIF_SOURCE_28, ITCN_B_TIM2_PAOVF_SOURCE_29, ITCN_B_PIT1_PIF_SOURCE_30, ITCN_B_PIT2_PIF_SOURCE_31, ITCN_B_EDGEPORT_EPF0_SOURCE_32, ITCN_B_EDGEPORT_EPF1_SOURCE_33, ITCN_B_EDGEPORT_EPF2_SOURCE_34, ITCN_B_EDGEPORT_EPF3_SOURCE_35, ITCN_B_EDGEPORT_EPF4_SOURCE_36, ITCN_B_EDGEPORT_EPF5_SOURCE_37, ITCN_B_EDGEPORT_EPF6_SOURCE_38, ITCN_B_EDGEPORT_EPF7_SOURCE_39
ITCN_B_Autovector_t	vectorType	Specify vectored or autovectored interrupt.	ITCN_B_VECTORED_INTERRUPTS, ITCN_B_AUTOVECTORED_INTERRUPTS

9.4.3.4 Return Values

Error Code	Description
ITCN_B_ERR_NONE	No error
ITCN_B_ERR_INVALID_HANDLE	ITCN base address parameter is zero
ITCN_B_ERR_INVALID_PRIORITY_LEVEL	Invalid priority level.
ITCN_B_ERR_INVALID_INTERRUPT_TYPE	Invalid interrupt type (fast or normal).
ITCN_B_ERR_INVALID_ISR_ADDRESS	Address to ISR is zero.
ITCN_B_ERR_INVALID_INTERRUPT_SOURCE	Interrupt source is not valid.
ITCN_B_ERR_INVALID_VECTOR_TYPE	Invalid vector type (autovectored or vectored interrupt).

9.4.4 ITCN_B_ControlOperation

9.4.4.1 Description

The ITCN_B_ControlOperation function enables or disables the interrupts at a given priority level. Parameter checking is a compile time option.

9.4.4.2 Prototype

```
ITCN_B_ReturnCode_t ITCN_B_ControlOperation (
          pITCN_B_t                   ITCNPtr,
          UINT8                       priorityLevel,
          ITCN_B_InterruptType_t      interruptType,
          ITCN_B_InterruptControl_t   interruptControl
)
```

9.4.4.3 Arguments

Data Type	Formal Name	Description	Valid Values
pITCN_B_t	ITCNPtr	ITCN base address associated with this driver.	Any non-zero core processor data address.
UINT8	priorityLevel	Priority level for this interrupt.	0, 1, …, 31
ITCN_B_InterruptType_t	interruptType	Specify fast or normal interrupt.	ITCN_B_INTERRUPT_NORMAL, ITCN_B_INTERRUPT_FAST
ITCN_B_InterruptControl_t	interruptControl	Enable or disable interrupts	ITCN_B_INTERRUPT_DISABLE, ITCN_B_INTERRUPT_ENABLE

9.4.4.4 Return Values

Error Code	Description
ITCN_B_ERR_NONE	No error
ITCN_B_ERR_INVALID_HANDLE	ITCN base address parameter is zero
ITCN_B_ERR_INVALID_PRIORITY_LEVEL	Invalid priority level.
ITCN_B_ERR_INVALID_INTERRUPT_TYPE	Invalid interrupt type (fast or normal).
ITCN_B_ERR_INVALID_INTERRUPT_CONTROL	Interrupt control (enable or disable) is not valid.

9.4.5 ITCN_B_ForceInterrupt

9.4.5.1 Description

The ITCN_B_ForceInterrupt function forces or clears an interrupt condition on a specified source. Parameter checking is a compile-time option.

9.4.5.2 Prototype

```
ITCN_B_ReturnCode_t ITCN_B_ForceInterrupt (
          pITCN_B_t                    ITCNPtr,
          ITCN_B_InterruptSource_t     interruptSource,
          ITCN_B_Force_t               forceControl
)
```

9.4.5.3 Arguments

Data Type	Formal Name	Description	Valid Values
pITCN_B_t	ITCNPtr	ITCN base address associated with this driver.	Any non-zero core processor data address.
ITCN_B_InterruptSource_t	interruptSource	Source of interrupt.	Any enumeration of type ITCN_B_InterruptSource_t
ITCN_B_Force_t	forceControl	Specify force or clear of interrupt	ITCN_B_CLEAR_INTERRUPT, ITCN_B_FORCE_INTERRUPT

9.4.5.4 Return Values

Error Code	Description
ITCN_B_ERR_NONE	No error
ITCN_B_ERR_INVALID_HANDLE	ITCN base address parameter is zero
ITCN_B_ERR_INVALID_INTERRUPT_SOURCE	Interrupt source is not valid.
ITCN_B_ERR_INVALID_FORCE_CONTROL	Force control selection is invalid

9.4.6 ITCN_B_GetStatus

9.4.6.1 Description

The ITCN_B_GetStatus function returns the status of the ITCN. Parameter checking is a compile-time option.

9.4.6.2 Prototype

```
ITCN_B_ReturnCode_t ITCN_B_GetStatus (
          pITCN_B_t             ITCNPtr,
          pITCN_B_Status_t      statusPtr
)
```

9.4.6.3 Arguments

Data Type	Formal Name	Description	Valid Values
pITCN_B_t	ITCNPtr	ITCN base address associated with this driver.	Any non-zero core processor data address.
pITCN_B_Status_t	statusPtr	Result address for status structure.	Any non-zero core processor data address.

9.4.6.4 Return Values

Error Code	Description
ITCN_B_ERR_NONE	No error
ITCN_B_ERR_INVALID_HANDLE	ITCN base address parameter is zero
ITCN_B_ERR_BAD_RESULT_ADDR	Result Pointer is zero

9.4.7 ITCN_B_GetRegister

9.4.7.1 Description

The ITCN_B_GetRegister function returns unformatted data from selected ITCN registers. Parameter checking is a compile-time option.

9.4.7.2 Prototype

```
ITCN_B_ReturnCode_t ITCN_B_GetRegister (
          pITCN_B_t             ITCNPtr,
          ITCN_B_Register_t     registerSwitch,
          UINT32 *              registerPtr
)
```

9.4.7.3 Arguments

Data Type	Formal Name	Description	Valid Values
pITCN_B_t	ITCNPtr	ITCN_B base address associated with this driver.	Any non-zero core processor data address.
ITCN_B_Register_t	registerSwitch	Select among a sub-set of ITCN_B registers.	Any enumeration of ITCN_B_Register_t
UINT32 *	registerPtr	Result address for selected ITCN_B register data.	Any non-zero core processor data address.

9.4.7.4 Return Values

Error Code	Description
ITCN_B_ERR_NONE	No error
ITCN_B_ERR_INVALID_HANDLE	ITCN base address parameter is zero
ITCN_B_ERR_BAD_RESULT_ADDR	Result Pointer is zero
ITCN_B_ERR_INVALID_REGISTER	ITCN Register Selection switch is invalid

9.4.8 ITCN_B_SetRegister

9.4.8.1 Description

The ITCN_B_SetRegister function writes unformatted data to selected interrupt controller (ITCN) registers. Parameter checking is a compile-time option.

9.4.8.2 Prototype

```
ITCN_B_ReturnCode_t ITCN_B_SetRegister (
          pITCN_B_t             ITCNPtr,
          ITCN_B_Register_t     registerSelect,
          UINT32                registerVal
)
```

9.4.8.3 Arguments

Data Type	Formal Name	Description	Valid Values
pITCN_B_t	ITCNPtr	ITCN_B base address associated with this driver.	Any non-zero core processor data address.
ITCN_B_Register_t	registerSelect	Select among a sub-set of ITCN_B registers.	ITCN_B_ICR, ITCN_B_IFRH, ITCN_B_IFRL, ITCN_B_NIER ITCN_B_FIER, ITCN_B_PLSR0 through ITCN_B_PLSR39,
UINT32	registerVal	Register value for selected ITCN_B register data.	if registerSelect == ITCN_B_ICR: 0x0000 -- 0xFFFF; if registerSelect ==ITCN_B_IFRH, ITCN_B_IFRL, ITCN_B_NIER, or ITCN_B_FIER: 0x00000000 --0xFFFFFFFF; if registerSelect == ITCN_B_PLSR0 through ITCN_B_PLSR39: 0x00 -- 0xFF

9.4.8.4 Return Values

Error Code	Description
ITCN_B_ERR_NONE	No error
ITCN_B_ERR_INVALID_HANDLE	ITCN base address parameter is zero
ITCN_B_ERR_INVALID_REGISTER	ITCN register selection is invalid
ITCN_B_ERR_INVALID_REGISTER_VALUE	Register value is out of range for register selected

9.4.9 ITCN_B_Reset

9.4.9.1 Description

The ITCN_B_Reset function returns all ITCN registers to their power-on reset state. Parameter checking is a compile-time option.

9.4.9.2 Prototype

```
ITCN_B_ReturnCode_t ITCN_B_Reset (
            pITCN_B_t
)
```

9.4.9.3 Arguments

Data Type	Formal Name	Description	Valid Values
pITCN_B_t	ITCNPtr	ITCN_B base address associated with this driver.	Any non-zero core processor data address.

9.4.9.4 Return Values

Error Code	Description
ITCN_B_ERR_NONE	No error
ITCN_B_ERR_INVALID_HANDLE	ITCN base address parameter is zero

9.5 ITCN_B Example Application

The following describes the ITCN_B example application.

9.5.1 Description for Example Application

This program sets a vector base address, initializes the interrupt controller, initalizes the vector table, sets up custom ISRs, enables interrupts, sets processor status to enable interrupts, forces interrupts, and waits for the interrupt counter to increment. The program returns on its own at completion.

9.5.1.1 Example Application Code

```
/*------------------------------------------------------------------------
                Example Application for ITCN_B Module

File: itcnb_app.c

Purpose:
         This program executes an example application of the M-CORE
         Peripherals Library ITCN_B Level 1 driver. It is setup to
         operation on an M-CORE 2107 CMB board.

         This program will set the vector base address, initalize
         the interrupt controller, initialize the vector table to
         a default ISR, Setup a custom ISR, enable interrupts,
         set processor status to enable interrupts, force an
         interrupt, and wait for the custom ISR to execute.

         If an error occurs, the program will return the line number of
         where the failure occurred.

Time to Run:
         Less than one second.

General Setup:
         General hardware and software setup to run this application
         can be found in the 'readme_app.txt' file, located in the
         '<library>/app' directory.

Application Specific Setup:
         Development Board: M-CORE 2107 CMB board.

         Application requires processor to operate in supervisor mode
         with interrupts disabled and exceptions enabled.

(C) Copyright Motorola Inc, 2000.  All rights reserved.
------------------------------------------------------------------------*/

#include "itcn_b.h"
#include "core_b.h"
#include "plibdefs_mmc2107.h"
```

```
/*********************************************************************/
/* Macro Definitions                                                 */
/*********************************************************************/

#define HANDLE_ERROR(err) if (err != ITCN_B_ERR_NONE) return(__LINE__)

#define ITCN_B_EXAMPLE_APP_VECTOR_BASE_ADDR 0x800000

#define ITCN_B_INTERRUPT_PRIORITY_LEVEL 31

#undef ITCN_B_PARAM_CHECKING       /* do in case of external definition */
#define ITCN_B_PARAM_CHECKING 1    /* 1= enable for debug, 0 = disable  */

/*********************************************************************/
/* Function Prototypes                                               */
/*********************************************************************/
static void DefaultHandler(void);            /* handler for all vectors */
static void TransmitCompleteHandler(void);   /* handler for vector*/
static void DefaultHandlerAssy(void);
static void TransmitCompleteHandlerAssy(void);
int itcnb_example (void);
int main (void);

/*********************************************************************/
/* Global Variables                                                  */
/*********************************************************************/
UINT8 globalInterruptCounter = 0;

pITCN_B_t ITCNPtr = (pITCN_B_t)__MMC2107_ITCN;  /* ITCN base address */

/*********************************************************************/
/*  Function: main                                                   */
/*                                                                   */
/*  Purpose: Drive program operation                                 */
/*                                                                   */
/*  Inputs: none                                                     */
/*                                                                   */
/*  Outputs: status (if passed, 0; else line number of the failure)  */
/*********************************************************************/
int main ()
{
    int status;

    status = itcnb_example ( );

    return (status);
}
```

ITCN_B Level 1

```
/*********************************************************************/
/* Function: itcnb_example                                           */
/*                                                                   */
/* Purpose: Provide an example for the usage of the ITCN_B           */
/*                                                                   */
/* Inputs:  none                                                     */
/*                                                                   */
/* Outputs: status (if passed, 0; else line number of the failure)   */
/*********************************************************************/
int itcnb_example ()
{
/*-------------------------------------------------------------------*/
/* Local variable declarations                                       */
/*-------------------------------------------------------------------*/
   int status;                          /* device driver API return code */

   pVectorTable_t pVTable;              /* points to vector base address */
   UINT32 HValue;                                  /* handler address */

   pVTable = (void *)ITCN_B_EXAMPLE_APP_VECTOR_BASE_ADDR; /* vector base*/

/*-------------------------------------------------------------------*/
/* Set Vector Base address                                           */
/*-------------------------------------------------------------------*/

   /* Call CORE_B_SetVectorBase with valid parameters */
   status = CORE_B_SetVectorBase( pVTable );

   HANDLE_ERROR(status);

/*-------------------------------------------------------------------*/
/* Initalize Interrupt Controller                                    */
/*-------------------------------------------------------------------*/
    /* Call CORE_B_Init with valid parameters */
    status = ITCN_B_Init ( ITCNPtr,
                           ITCN_B_VECTORED_INTERRUPTS,
                           ITCN_B_FAST_INTERRUPT_DISABLE,
                           ITCN_B_MASK_DISABLE,
                           ITCN_B_INTERRUPT_PRIORITY_LEVEL );

   HANDLE_ERROR(status);

/*-------------------------------------------------------------------*/
/* Initalize Vector Table to default ISR                             */
/*-------------------------------------------------------------------*/

   HValue = (VectorValue_t)&DefaultHandler;

   /* Call ITCN_B_InitVectorTable with valid parameters */
   status = ITCN_B_InitVectorTable( ITCNPtr,
                                    HValue,
                                    NULL );

   HANDLE_ERROR(status);
```

```
/*-------------------------------------------------------------------------*/
/* Initalize Interrupt Vector for SCI Transmit Complete                    */
/*-------------------------------------------------------------------------*/

   HValue = (VectorValue_t)&TransmitCompleteHandler;

   /* Call ITCN_B_SetInterruptServiceRoutine with valid parameters */
   status = ITCN_B_SetInterruptServiceRoutine(
                                ITCNPtr,
                                ITCN_B_INTERRUPT_PRIORITY_LEVEL,
                                ITCN_B_INTERRUPT_NORMAL,
                                HValue,
                                ITCN_B_SCI1_TC_SOURCE_7,
                                ITCN_B_VECTORED_INTERRUPTS );

   HANDLE_ERROR(status);

/*-------------------------------------------------------------------------*/
/* Enable ITCN interrupts                                                  */
/*-------------------------------------------------------------------------*/

   /* Call ITCN_B_ControlOperation with valid parameters */
   status = ITCN_B_ControlOperation ( ITCNPtr,
                                      ITCN_B_INTERRUPT_PRIORITY_LEVEL,
                                      ITCN_B_INTERRUPT_NORMAL,
                                      ITCN_B_INTERRUPT_ENABLE );

   HANDLE_ERROR(status);

/*-------------------------------------------------------------------------*/
/* Set processor status to enable interrupts                               */
/*-------------------------------------------------------------------------*/

   /* Call CORE_B_SetProcessorStatus with valid parameters */
   status = CORE_B_SetProcessorStatus (
                  CORE_B_TRACE_NORMAL,
               CORE_B_EXCEPTION_ENABLE,        /* enable exceptions */
               CORE_B_INTERRUPT_ON_INSTRUCTION_BOUNDARIES,
               CORE_B_INTERRUPT_ENABLE_ONLY_NORMAL );

   HANDLE_ERROR(status);

/*-------------------------------------------------------------------------*/
/* Force SCI Transmit Complete Interrupt                                   */
/*-------------------------------------------------------------------------*/
    /* force interrupt */
   status = ITCN_B_ForceInterrupt ( ITCNPtr,
                                    ITCN_B_SCI1_TC_SOURCE_7,
                                    ITCN_B_FORCE_INTERRUPT );

   HANDLE_ERROR(status);
```

ITCN_B Level 1

```
/*-------------------------------------------------------------------------*/
/* Loop until GlobalInterruptCounter is equal to one                       */
/*-------------------------------------------------------------------------*/
   while(globalInterruptCounter == 0)
   {
    ; /* wait for exception handler to increment globalInterruptCounter */
   }

/*-------------------------------------------------------------------------*/
/* Disable ITCN interrupts                                                 */
/*-------------------------------------------------------------------------*/

   /* Call ITCN_B_ControlOperation with valid parameters */
   status = ITCN_B_ControlOperation ( ITCNPtr,
                                      ITCN_B_INTERRUPT_PRIORITY_LEVEL,
                                      ITCN_B_INTERRUPT_NORMAL,
                                      ITCN_B_INTERRUPT_DISABLE );

   HANDLE_ERROR(status);


/*-------------------------------------------------------------------------*/
/* Return ITCN_B_ERR_NONE if no execution errors occur                     */
/*-------------------------------------------------------------------------*/
   return(0);
}

/***************************************************************************/
/* Function: TransmitCompleteHandler                                       */
/*                                                                         */
/* Purpose: Interrupt Service Routine for Transmit Complete Interrupt      */
/*                                                                         */
/* Inputs:  none                                                           */
/*                                                                         */
/* Outputs: none                                                           */
/***************************************************************************/
static void TransmitCompleteHandler (void)
{
    globalInterruptCounter ++;  /* increment interrupt counter */

    /* clear forced interrupt */
    /* note:  if interrupt was not cleared here, we could not break out */
    /*        of this Interrupt Service Routine */
    ITCN_B_ForceInterrupt ( ITCNPtr,
                            ITCN_B_SCI1_TC_SOURCE_7,
                           ITCN_B_CLEAR_INTERRUPT );

    TransmitCompleteHandlerAssy();
        /* clear interrupt */
}
```

```
/*********************************************************************/
/* Function: DefaultHandler                                          */
/*                                                                   */
/* Purpose: Default Interrupt Service Routine                        */
/*                                                                   */
/* Inputs:  none                                                     */
/*                                                                   */
/* Outputs: none                                                     */
/*********************************************************************/

static void DefaultHandler (void)
{
    globalInterruptCounter ++;  /* increment interrupt counter */
    DefaultHandlerAssy();
}
      .export     DefaultHandlerAssy
      .export     TransmitCompleteHandlerAssy

      .text

/* internal functions */
/*********************************************************************/
/* Function: DefaultHandler (assy used can compile with CodeWarrior) */
/*                                                                   */
/* Purpose: Vector here on unexpected exceptions while running main  */
/*                                                                   */
/* Inputs & Outputs:  none                                           */
/*********************************************************************/
DefaultHandlerAssy:
  subi r0,32                /* advance stack pointer */
  stm r1-r15,(r0)           /* save register context */
  bkpt                      /* breakpoint for debugging */
  ldm r1-r15,(r0)           /* restore register context */
  addi r0,32                /* restore stack pointer */
  rte                       /* return from exception */

/*********************************************************************/
/* Function: DivideByZeroHandler (assy used can compile w/ CodeWarrior) */
/*                                                                   */
/* Purpose: Vector here on divide by zero exception while running main  */
/*                                                                   */
/* Inputs & Outputs:  none                                           */
/*********************************************************************/
TransmitCompleteHandlerAssy:
  subi r0,32                /* advance stack pointer */
  stm r1-r15,(r0)           /* save register context */
  bkpt                      /* breakpoint for debugging */
  mfcr r2,epc               /* get the exception program counter */
  addi r2,2                 /* advance to instruction after divide */
  mtcr r2,epc               /* to continue prog after rte */
  ldm r1-r15,(r0)           /* restore register context */
  addi r0,32                /* restore stack pointer */
  rte                       /* return from exception */
```

Section 10 MOTO_FLASH_A Level 1

This section describes the MOTO_FLASH_A Level 1 M•CORE™ Device Driver.

10.1 Overview

The CDR MoneT FLASH EEPROM for RLB (CMFR) Flash and its accompanying driver (MOTO_FLASH_A) are used on the MMC2107 development system. The level 1 service is designed to allow the programmer to control the CMFR on a near-register level, but without the concern of exact positions of registers and fields, or the detailed sequences of register operations. Refer to the MMC2107 Advance Information (Data Book) (MMC2107/D) for additional information.

10.2 MOTO_FLASH_A Service Functions

The MOTO_FLASH_A device driver software provides the following functions:

- Initializes the flash to a user defined state
- Enables or disables the module's low power mode based on user input
- Allows or disallows the module's debug mode based on user input
- Enables or diables the module's emulation mode based on user input
- Enables or disables access to the module's shadow information based on user input.
- Locks or unlocks the module's configuration bits for modification based on user input
- Enables or disables all array accesses, depening on user input
- Configures the module's boot mode after reset
- Configures the block or blocks specified for three protection modes, depending on user input
- Configures the CLKPM, CLKPE, and SCLKR fields of the CMFRCTL register in order to set the specified pulse width of the specified program/erase cycle for the specified value of the system clock
- Gets the status of a programming or erase pulse
- Select single or multiple blocks for erasing or programming, depending on user input
- Gets the status of high voltage operations
- Configures the array write mode to program or erase, depending on user input
- Starts or ends a program/erase sequence, depending on user input
- Enables or disables a program/erase pulse, depending on user input
- Gets the contents of the specified CMFR Flash register
- Writes a 32-bit data value to the specified CMFR Flash register
- Erases multiple blocks within a single flash array or its shadow region based on user input

- Erases multiple blocks within multiple flash arrays and their shadow regions based on user input
- Programs multiple blocks within a single flash array or its shadow region based on user input
- Checks the array or shadow information for blank content

10.3 MOTO_FLASH_A Data Type Definitions

The following paragraphs show standard, board specific, abstract and enumerated data type definitions.

10.3.1 Standard Data Type Definitions

- UINT8 — 8-bit unsigned data value
- INT8 — 8-bit signed data value
- UINT16 — 16-bit unsigned data value
- INT16 — 16-bit signed data value
- UINT32 — 32-bit unsigned data value
- INT32 — 32-bit signed data value
- ddBoolean_t — boolean data value (32 bits)

10.3.2 Board Specific Base Address Definitions

```
#define __SIKA_CMFR_BASE_ADDRESS  (pMOTO_FLASH_A_t(0x1000000))
```

10.3.3 Abstract Data Type Definitions

10.3.3.1 MOTO_FLASH_A_t – CMFR Flash Register Definition

```
typedef struct {
      volatile UINT32 CMFRMCR;      /* CMFR Module Configuration Register */
      volatile UINT32 CMFRMTR;      /* CMFR Module Test Register */
      volatile UINT32  CMFRCTL;     /* CMFR High Voltage Control Register */
} MOTO_FLASH_A_t, *pMOTO_FLASH_A_t;
```

10.3.3.2 MOTO_FLASH_A_Data_t – CMFR Data Type

```
typedef UINT32 MOTO_FLASH_A_Data_t;
```

10.3.3.3 MOTO_FLASH_A_Address_t – CMFR Address Type

```
typedef MOTO_FLASH_A_Data_t *pMOTO_FLASH_A_Address_t;
```

10.3.3.4 MOTO_FLASH_A_PulseWidth_t – CMFR Pulse Width Data Type

```
typedef UINT32 MOTO_FLASH_A_PulseWidth_t;    /* units in microseconds  *
```

10.3.3.5 MOTO_FLASH_A_SystemClock_t – CMFR System Clock Data Type

```
typedef UINT32 MOTO_FLASH_A_SystemClock_t; /* Units in megahertz */
```

10.3.4 Enumerated Data Type Definitions

10.3.4.1 MOTO_FLASH_A_RegisterSwitch_t – MOTO Flash Register Selection

```
typedef enum {
        MOTO_FLASH_A_CMFRMCR_SWITCH,        /* Select CMFR Module Configuration
Register */
        MOTO_FLASH_A_CMFRMTR_SWITCH,        /* Select CMFR Module Test Register */
        MOTO_FLASH_A_CMFRCTL_SWITCH         /* Select CMFR Module Control Register*/
} MOTO_FLASH_A_RegisterSwitch_t;
```

10.3.4.2 MOTO_FLASH_A_Control_t - MOTO_FLASH_A Action Selection

```
typedef enum {
        MOTO_FLASH_A_ENABLE,
        MOTO_FLASH_A_DISABLE
}MOTO_FLASH_A_Control_t;
```

10.3.4.3 MOTO_FLASH_A_ShadowBlock_t - MOTO_FLASH_A Shadow Block Selection (value sensitive)

```
typedef enum {
        MOTO_FLASH_A_LOW_FILL_SHADOW_BLOCK = 0,
        MOTO_FLASH_A_HIGH_FILL_SHADOW_BLOCK = 2
}MOTO_FLASH_A_ShadowBlock_t;
```

10.3.4.4 MOTO_FLASH_A_ShadowStartOffset_t – MOTO_FLASH_A Shadow Start Offset Selection

```
typedef enum {
        MOTO_FLASH_A_SHADOW_START_OFFSET_0_BYTES = 0,
        MOTO_FLASH_A_SHADOW_START_OFFSET_16_BYTES = 0,
} MOTO_FLASH_A_ShadowStartOffset_t;
```

10.3.4.5 MOTO_FLASH_A_ShadowEndOffset_t – MOTO_FLASH_A Shadow End Offset Selection

```
typedef enum {
        MOTO_FLASH_A_SHADOW_END_OFFSET_256_BYTES
} MOTO_FLASH_A_ShadowEndOffset_t;
```

10.3.4.6 MOTO_FLASH_A_DataWidth_t - MOTO_FLASH_A DataWidth Selection

```
typedef enum {
        MOTO_FLASH_A_2_BYTES,               /* 16bit memory data width */
        MOTO_FLASH_A_4_BYTES                /* 32bit memory data width */
}MOTO_FLASH_A_DataWidth_t;
```

10.3.4.7 MOTO_FLASH_A_BlockSize_t - MOTO_FLASH_A BlockSize Selection (value sensitive)

```
typedef enum {
         MOTO_FLASH_A_16K_BYTES = 1<<14,     /* 16K byte memory block length */
         MOTO_FLASH_A_32K_BYTES = 1<<15      /* 32K byte memory block length */
}MOTO_FLASH_A_BlockSize_t
```

10.3.4.8 MOTO_FLASH_A_Lock_t - MOTO_FLASH_A Lock Selection

```
typedef enum {
         MOTO_FLASH_A_LOCK,
         MOTO_FLASH_A_UNLOCK
}MOTO_FLASH_A_Lock_t;
```

10.3.4.9 MOTO_FLASH_A_CycleType_t - MOTO_FLASH_A Cycle Type Selection

```
typedef enum {
         MOTO_FLASH_A_PROGRAM,
         MOTO_FLASH_A_ERASE,
         MOTO_FLASH_A_READ
}MOTO_FLASH_A_CycleType_t;
```

10.3.4.10 MOTO_FLASH_A_Sequence_t - MOTO_FLASH_A Sequence Start or End Selection

```
typedef enum {
         MOTO_FLASH_A_START,
         MOTO_FLASH_A_END
}MOTO_FLASH_A_Sequence_t;
```

10.3.4.11 MOTO_FLASH_A_BlockSelect_t - MOTO_FLASH_A Block Selection (value sensitve)

```
typedef enum {
         MOTO_FLASH_A_NO_BLOCK_SELECTED = 0x00,
         MOTO_FLASH_A_BLOCK_0 = 0x01,
         MOTO_FLASH_A_BLOCK_1 = 0x02,
         MOTO_FLASH_A_BLOCK_2 = 0x04,
         MOTO_FLASH_A_BLOCK_3 = 0x08,
         MOTO_FLASH_A_BLOCK_4 = 0x10,
         MOTO_FLASH_A_BLOCK_5 = 0x20,
         MOTO_FLASH_A_BLOCK_6 = 0x40,
         MOTO_FLASH_A_BLOCK_7 = 0x80
}MOTO_FLASH_A_BlockSelect_t;
```

10.3.4.12 MOTO_FLASH_A_PrivilegeProtection_t - MOTO_FLASH_A Privilege Protection

```
typedef enum {
         MOTO_FLASH_A_USER_ACCESS,
         MOTO_FLASH_A_SUPERVISOR_ACCESS,
         MOTO_FLASH_A_NO_PRIVILEDGE_CHANGE
```

```
} MOTO_FLASH_A_PrivilegeProtection_t;
```

10.3.4.13 MOTO_FLASH_A_TextDataProtection_t – MOTO_FLASH_A Code/Text Protection

```
typedef enum {
         MOTO_FLASH_A_DATA_ACCESS,
         MOTO_FLASH_A_TEXT_AND_DATA_ACCESS,
         MOTO_FLASH_A_NO_TEXT_DATA_PROTECTION_CHANGE
} MOTO_FLASH_A_TextDataProtection_t;
```

10.3.4.14 MOTO_FLASH_A_ReadWriteProtection_t - Read/Write Protection

```
typedef enum {
         MOTO_FLASH_A_READ_ONLY,
         MOTO_FLASH_A_READ_WRITE,
         MOTO_FLASH_A_NO_READ_WRITE_PROTECTION_CHANGE
} MOTO_FLASH_A_ReadWriteProtection_t;
```

10.3.5 Register Macros

10.3.5.1 CMFR Module Configuration Register (CMFRMCR)

```
#define CMFR_BLOCK_FIELD_MAX 255
#define CMFRMCR_PROTECT_BITNO 0           /* Read Only/Write Blk Protection  */
#define CMFRMCR_PROTECT_MASK (CMFR_BLOCK_FIELD_MAX<<CMFRMCR_PROTECT_BITNO)
#define CMFRMCR_DATA_BITNO 8              /* Text/DataText Blk Protection  */
#define CMFRMCR_DATA_MASK (CMFR_BLOCK_FIELD_MAX << CMFRMCR_DATA_BITNO)
#define CMFRMCR_SUPV_BITNO 16             /* Supervisor/User Blk Protection  */
#define CMFRMCR_SUPV_MASK (CMFR_BLOCK_FIELD_MAX << CMFRMCR_SUPV_BITNO)
#define CMFRMCR_BOOT_BITNO 24             /* Bootstrap Enable */
#define CMFRMCR_BOOT_MASK (1<<CMFRMCR_BOOT_BITNO)
#define CMFRMCR_DIS_BITNO 25              /* Flash Information Disable */
#define CMFRMCR_DIS_MASK (1<<CMFRMCR_DIS_BITNO)
#define CMFRMCR_LOCK_BITNO 26             /* Lock Control */
#define CMFRMCR_LOCK_MASK (1<<CMFRMCR_LOCK_BITNO)
#define CMFRMCR_SIE_BITNO 27              /* Lock Control */
#define CMFRMCR_SIE_MASK (1<<CMFRMCR_SIE_BITNO)
#define CMFRMCR_EME_BITNO 28              /* Emulation Enable */
#define CMFRMCR_EME_MASK ( 1<<CMFRMCR_EME_BITNO)
#define CMFRMCR_FRZ_BITNO 30              /* Freeze Enable */
#define CMFRMCR_FRZ_MASK (1<<CMFRMCR_FRZ_BITNO)
#define CMFRMCR_STOP_BITNO 3              /* Stop Enable */
#define CMFRMCR_STOP_MASK (1<<CMFRMCR_STOP_BITNO)
```

10.3.5.2 CMFR Module Test Register (CMFRMTR)

```
#define CMFRMTR_PAWS_BITNO 8  /* Program Amplitude/Width Modulation Select */
#define CMFRMTR_PAWS_MAX 7
#define CMFRMTR_PAWS_MASK (CMFRMTR_PAWS_MAX << CMFRMCR_PAWS_BITNO)
```

10.3.5.3 CMFR High Voltage Control Register (CMFRCTL)

```
#define CMFRCTL_EHV_BITNO 0          /* Enable a Program or Erase Pulse */
#define CMFRCTL_EHV_MASK  (1<<CMFRCTL_EHV_BITNO)
#define CMFRCTL_SES_BITNO 1          /* Start-End a Program or Erase Sequence */
#define CMFRCTL_SES_MASK  (1<<CMFRCTL_SES_BITNO)
#define CMFRCTL_PE_BITNO 2           /* Program or Erase Select */
#define CMFRCTL_PE_MASK (1<<CMFRCTL_PE_BITNO)
#define CMFRCTL_PEEM_BITNO 5         /* Program Erase Enable Monitor (ReadOnly)*/
#define CMFRCTL_PEEM_MASK (1<<CMFRCTL_PEEM_BITNO)
#define CMFRCTL_BLOCK_BITNO  8       /* Block Program and Erase Select  */
#define CMFRCTL_BLOCK_MASK (CMFR_BLOCK_FIELD_MAX << CMFRCTL_BLOCK_BITNO)
#define CMFRCTL_CLKPM_BITNO 16       /* Clock Period Multiple */
#define CMFRCTL_CLKPM_MAX 127
#define CMFRCTL_CLKPM_MASK (CMFRCTL_CLKPM_MAX <<CMFRCTL_CLKPM_BITNO)
#define CMFRCTL_CLKPE_BITNO 24       /* Clock Period Exponent */
#define CMFRCTL_CLKPE_MAX 3
#define CMFRCTL_CLKPE_MASK (CMFRCTL_CLKPE_MAX <<CMFRCTL_CLKPE_BITNO)
#define CMFRCTL_SCLKR_BITNO 27       /* Clock Period Exponent */
#define CMFRCTL_SCLKR_MAX  7
#define CMFRCTL_SCLKR_MASK (CMFRCTL_SCLKR_MAX <<CMFRCTL_SCLKR_BITNO)
#define CMFRCTL_HVS_BITNO 31         /* High Voltage Pulse Status */
#define CMFRCTL_HVS_MASK (1<<CMFRCTL_HVS_BITNO)
```

10.3.5.4 Miscellaneous Constants

```
/* Maximum and minimum system clock frequency */
#define CMFR_SYSTEM_CLOCK_MIN_FREQ  MOTO_FLASH_A_8_MHZ      /* 8 MHZ */
#define CMFR_SYSTEM_CLOCK_MAX_FREQ  MOTO_FLASH_A_40_MHZ     /* 40 MHZ */
```

10.3.6 Return Codes

Return codes from the enumerated type “ddErr_t”.

Return Code	Functions Returning	Description
DD_ERR_NONE	All	No error.
DD_ERR_INVALID_HANDLE	All except BlankCheck	Moto_Flash base address parameter is zero.
DD_ERR_BAD_RESULT_ADDR	GetRegister,GetPulseStatus, GetHighVoltageStatus	Result Pointer is zero.
DD_ERR_INVALID_ADDRESS	EraseSerialMRVerify, BlankCheck	Data Pointer is NULL.
DD_ERR_INVALID_REGISTER	GetRegister, SetRegister	
MOTO_FLASH_A_ERR_HIGH_VOLTAGE_OPERATION_DISABLED	Init	High voltage operation is disabled.
MOTO_FLASH_A_ERR_INVALID_BLOCK_CONFIG	SelectBlocks, Init, ControlBlockProtection, EraseSerialMRVerify, EraseParallelMRVerify, ProgramSerialMRVerify	The block configuration specified is not in range

Return Code	Functions Returning	Description
MOTO_FLASH_A_ERR_INVALID_BLOCK_SIZE	EraseSerialMRVerify, EraseParallelMRVerify, ProgramSerialMRVerify	The block size specified is not in range
MOTO_FLASH_A_ERR_INVALID_SYSTEM_CLOCK	SetPulseWidth, ControlBootMode, Init	The system clock value specified is out of range
MOTO_FLASH_A_ERR_INVALID_PULSE_WIDTH	SetPulseWidth, ControlBootMode, Init	The pulse width value specified is out of range
MOTO_FLASH_A_ERR_PULSE_ACTIVE	SetPulseWidth	The program/erase pulse is active
MOTO_FLASH_A_ERR_INVALID_DATA_WIDTH	EraseSerialMRVerify, EraseParallelMRVerify, ProgramSerialMRVerify, BlankCheck	The pulse width value specified is out of range
MOTO_FLASH_A_ERR_INVALID_CYCLE_TYPE	SetPulseWidth, Init	The cycle type value specified is out of range
MOTO_FLASH_A_ERR_INVALID_PROTECTION	ControlBlockProtection	The protection value specified is out of range
MOTO_FLASH_A_ERR_INVALID_SHADOW_BLOCK	EraseSerialMRVerify, EraseParallelMRVerify,	Specified shadow block value is out of range
MOTO_FLASH_A_ERR_INVALID_SHADOW_OFFSET	EraseSerialMRVerify, EraseParallelMRVerify	Specified shadow offset value is out of range
MOTO_FLASH_A_ERR_INVALID_MAX_PULSES	EraseSerialMRVerify, EraseParallelMRVerify, ProgramSerialMRVerify	Specified max pulses value is out of range
MOTO_FLASH_A_ERR_INVALID_DEVICE_COUNT	EraseParallelMRVerify	Specified device count is out of range
MOTO_FLASH_A_ERR_ARRAY_ERASE_FAILURE	EraseSerialMRVerify, EraseParallelMRVerify	MRVerifyFailed on Array and Max number of erase pulses was reached
MOTO_FLASH_A_ERR_SHADOW_ERASE_FAILURE	EraseSerialMRVerify, EraseParallelMRVerify	MRVerifyFailed on Shadow and Max number of erase pulses was reached
MOTO_FLASH_A_ERR_PROGRAM_FAILURE	ProgramSerialMRVerify	Failure to program array within specified max number of pulses
MOTO_FLASH_A_ERR_DEVICE_NOT_BLANK	BlankCheck	Device is not blank in the address range specified
MOTO_FLASH_A_ERR_PROGRAMMING_FUSES	ControlBootMode	Could not reprogram reset values of BOOT or DIS bits in the CMFRMCR register
MOTO_FLASH_A_ERR_INVALID_MODE	ControlLowPowerMode, ControlDebugMode, ControlEmulationMode, ControlShadowAccess, ControlWriteLock, ControlArrayAccess, ControlSequence, ControlPulse	Specified mode control is invalid

10.4 MOTO_FLASH_A API Definitions

10.4.1 MOTO_FLASH_A_Init

10.4.1.1 Description

The MOTO_FLASH_A_Init function initializes the flash to a user defined state. Parameter checking is a compile-time option.

10.4.1.2 Prototype

```
ddErr_t MOTO_FLASH_A_Init (
         pMOTO_FLASH_A_t                      Moto_FLASHPtr,
         MOTO_FLASH_A_CycleType_t             CycleType,
         MOTO_FLASH_A_BlockSelect_t           BlocksToWrite,
         MOTO_FLASH_A_Control_t               ShadowAccess,
         MOTO_FLASH_A_SystemClock_t           SystemClock,
         MOTO_FLASH_A_PulseWidth_t            PulseWidth,
         MOTO_FLASH_A_BlockSelect_t           BlocksToProtect,
         MOTO_FLASH_A_ReadWriteProtection_t   ReadWriteProtection,
         MOTO_FLASH_A_PrivilegeProtection_t   PrivilegeProtection,
         MOTO_FLASH_A_TextDataProtection_t    TextDataProtection,
         )
```

10.4.1.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	Moto_FLASHPtr	MOTO_FLASH_A base address associated with this driver.	Any non-zero core processor data address.
MOTO_FLASH_A_CycleType_t	CycleType	Determines type of flash memory cycle	Any enumerated value of MOTO_FLASH_A_CycleType_t
MOTO_FLASH_A_BlockSelect_t	BlocksToWrite	Determines which block or blocks are selected for write operations	Any enumerated value of MOTO_FLASH_A_BlockSelect_t, or bitwise or of its enumerated values
MOTO_FLASH_A_BlockSelect_t	BlocksToProtect	Determines which block or blocks are selected for protection	Any enumerated value of MOTO_FLASH_A_BlockSelect_t, or bitwise or of its enumerated values
MOTO_FLASH_A_PrivilegeProtection_t	PrivilegeProtection	Specifies whether block can be accessed exclusively by supervisor, or by user and supervisor	Any enumerated value of MOTO_FLASH_A_PrivilegeProtection_t
MOTO_FLASH_A_ReadWriteProtection_t	ReadWriteProtection	Specifies whether block can be written, or is read-only	Any enumerated value of MOTO_FLASH_A_ReadWriteProtection_t

MOTO_FLASH_A_TextDataProtection_t	TextDataProtection	Specifies whether block can be accessed by text accesses, or by text and data accesses	Any enumerated value of MOTO_FLASH_A_TextDataProtection_t
MOTO_FLASH_A_Control_t	ShadowAccess	Determines whether shadow information is accessed	Any enumerated value of MOTO_FLASH_A_Control_t
MOTO_FLASH_A_PulseWidth_t	PulseWidth	Desired pulse width in units of microseconds	If CycleType = MOTO_FLASH_A_PROGRAM: 21.2uS to 32uS If CycleType = MOTO_FLASH_A_ERASE: 0.9S to 1.1S if CycleType = READ: NULL
MOTO_FLASH_A_SystemClock_t	SystemClock	System Clock value in units of HZ	If CycleType = MOTO_FLASH_A_PROGRAM or CycleType = MOTO_FLASH_A_ERASE: 8 to 40 MHZ if CycleType = MOTO_FLASH_A_READ: NULL

10.4.1.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
MOTO_FLASH_A_ERR_HIGH_VOLTAGE_OPERAATION_DISABLED	High voltage operations are physically disabled
MOTO_FLASH_A_ERR_INVALID_BLOCK_CONFIG	Specified block config value is out of range
MOTO_FLASH_A_ERR_INVALID_SYSTEM_CLOCK	Specified system clock value is out of range
MOTO_FLASH_A_ERR_INVALID_PULSE_WIDTH	Specified pulse width value is out of range
MOTO_FLASH_A_ERR_INVALID_CYCLE_TYPE	Specified cycle type is out of range
MOTO_FLASH_A_ERR_INVALID_PROTECTION	Specified protection mode is out of range

10.4.2 MOTO_FLASH_A_ControlLowPowerMode

10.4.2.1 Description

MOTO_FLASH_A_ControlLowPowerMode enables or disable the module’s low power mode based on user input. Parameter checking is a compile-time option.

10.4.2.2 Prototype

```
ddErr_t MOTO_FLASH_A_ControlLowPowerMode (
         pMOTO_FLASH_A_t            FLASHPtr,
         MOTO_FLASH_A_Control_t     ModeControl
)
```

10.4.2.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
MOTO_FLASH_A_Control_t	ModeControl	Determines low power mode of operation	MOTO_FLASH_A_ENABLE = Enter low power mode MOTO_FLASH_A_DISABLE = Exit low power mode

10.4.2.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
MOTO_FLASH_A_ERR_INVALID_MODE	Specified mode control is invalid

10.4.3 MOTO_FLASH_A_ControlDebugMode

10.4.3.1 Description

MOTO_FLASH_A_ControlDebugMode allows or disallows entering the module's debug mode based on user input. Parameter checking is a compile-time option.

10.4.3.2 Prototype

```
ddErr_t MOTO_FLASH_A_ControlDebugMode (
         pMOTO_FLASH_A_t            FLASHPtr,
         MOTO_FLASH_A_Control_t     ModeControl
         )
```

10.4.3.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.

MOTO_FLASH_A_Control_t	ModeControl	Determines debug mode of operation	MOTO_FLASH_A_ENABLE = Allow debug mode when rlb_dbug_b is asserted MOTO_FLASH_A_DISABLE = Ignore debug mode when rlb_dbug_b is asserted

10.4.3.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
MOTO_FLASH_A_ERR_INVALID_MODE	Specified mode control is invalid

10.4.4 MOTO_FLASH_A_ControlEmulationMode

10.4.4.1 Description

MOTO_FLASH_A_ControlEmulationMode enables or disables the module's emulation mode based on user input. Parameter checking is a compile-time option.

10.4.4.2 Prototype

```
ddErr_t MOTO_FLASH_A_ControlEmulationMode (
         pMOTO_FLASH_A_t            FLASHPtr,
         MOTO_FLASH_A_Control_t     ModeControl
         )
```

10.4.4.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
MOTO_FLASH_A_Control_t	ModeControl	Determines emulation mode of operation	MOTO_FLASH_A_ENABLE = Enter emulation mode MOTO_FLASH_A_DISABLE = Enter normal mode

10.4.4.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
MOTO_FLASH_A_ERR_INVALID_MODE	Control mode specified is invalid

10.4.5 MOTO_FLASH_A_ControlShadowAccess

10.4.5.1 Description

MOTO_FLASH_A_ControlShadowAccess enables or disables access to the module's shadow information based on user input. When access to the shadow information is disabled, normal array accesses are enabled. Parameter checking is a compile-time option.

10.4.5.2 Prototype

```
ddErr_t MOTO_FLASH_A_ControlShadowAccess (
         pMOTO_FLASH_A_t            FLASHPtr,
         MOTO_FLASH_A_Control_t     AccessControl
         )
```

10.4.5.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
MOTO_FLASH_A_Control_t	AccessControl	Determines shadow access mode of operation	MOTO_FLASH_A_ENABLE = Enable shadow access MOTO_FLASH_A_DISABLE = Disable shadow access

10.4.5.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
MOTO_FLASH_A_ERR_INVALID_MODE	Mode control input is invalid

10.4.6 MOTO_FLASH_A_ControlWriteLock

10.4.6.1 Description

MOTO_FLASH_A_ControlWriteLock locks or unlocks the module's configuration bits for modification based on user input. Parameter checking is a compile-time option.

10.4.6.2 Prototype

```
ddErr_t MOTO_FLASH_A_ControlWriteLock (
         pMOTO_FLASH_A_t            FLASHPtr,
         MOTO_FLASH_A_Lock_t        LockControl
         )
```

10.4.6.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
MOTO_FLASH_A_Control_t	LockControl	Determines write lock mode of operation	MOTO_FLASH_A_UNLOCK= Unlock bits for modification MOTO_FLASH_A_LOCK= Lock bits, preventing their modification

10.4.6.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
MOTO_FLASH_A_ERR_INVALID_MODE	Mode control input is invalid

10.4.7 MOTO_FLASH_A_ControlArrayAccess

10.4.7.1 Description

MOTO_FLASH_A _ControlArrayAccess enables or disables all array accesses, depending on user input. Parameter checking is a compile-time option.

10.4.7.2 Prototype

```
DdErr_t MOTO_FLASH_A_ControlArrayAccess (
         pMOTO_FLASH_A_t            FLASHPtr,
         MOTO_FLASH_A_Control_t     AccessControl
         )
```

10.4.7.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
MOTO_FLASH_A_Control_t	AccessControl	Determines array access mode of operation	MOTO_FLASH_A_ENABLE= enable array accesses MOTO_FLASH_A_DISABLE = disable array accesses

10.4.7.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
MOTO_FLASH_A_ERR_INVALID_MODE	Mode control input is invalid

10.4.8 MOTO_FLASH_A_ControlBootMode

10.4.8.1 Description

MOTO_FLASH_A_ControlBootMode configures the module's boot mode after reset. The driver first erases the BOOT and DIS fuses, and then programs the fuses based on user-supplied values. Parameter checking is a compile-time option.

10.4.8.2 Prototype

```
ddErr_t MOTO_FLASH_A_ControlBootMode (
         pMOTO_FLASH_A_t                    FLASHPtr,
         MOTO_FLASH_A_Control_t             BOOTResetValue,
         MOTO_FLASH_A_Control_t             DISResetValue,
         MOTO_FLASH_A_PulseWidth_t          PulseWidth,
         MOTO_FLASH_A_SystemClock_t         SystemClock
         )
```

10.4.8.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
MOTO_FLASH_A_Control_t	BOOTResetValue	Determines reset value of BOOT bit of the CMFRMCR register	Any enumerated value of MOTO_FLASH_A_Control_t
MOTO_FLASH_A_Control_t	DISResetValue	Determines reset value of DIS bit of the CMFRMCR register	Any enumerated value of MOTO_FLASH_A_Control_t
MOTO_FLASH_A_SystemClock_t	SystemClock	System Clock value in units of HZ	If CycleType = MOTO_FLASH_A_PROGRAM or MOTO_FLASH_A_ERASE: 8 to 40 MHZ if CycleType = MOTO_FLASH_A_READ: NULL
MOTO_FLASH_A_PulseWidth_t	PulseWidth	Desired pulse width in units of microseconds	If CycleType = MOTO_FLASH_A_PROGRAM: 21.2uS to 32uS If CycleType = MOTO_FLASH_A_ERASE: 0.9S to 1.1S if CycleType = READ: NULL

10.4.8.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
MOTO_FLASH_A_ERR_INVALID_PULSE_WIDTH	Pulse width specified is out of range
MOTO_FLASH_A_ERR_INVALID_SYSTEM_CLOCK	System clock specified is out of range
MOTO_FLASH_A_ERR_PROGRAMMING_FUSES	Error programming fuses

10.4.9 MOTO_FLASH_A_ControlBlockProtection

10.4.9.1 Description

MOTO_FLASH_A _ControlBlockProtection configures the block or blocks specified for three protection modes, depending on user input. Parameter checking is a compile-time option.

10.4.9.2 Prototype

```
ddErr_t MOTO_FLASH_A_ControlBlockProtection (
         pMOTO_FLASH_A_t                      FLASHPtr,
         MOTO_FLASH_A_BlockSelect_t           BlocksToProtect,
         MOTO_FLASH_A_ReadWriteProtection_t   ReadWriteProtection,
         MOTO_FLASH_A_PrivilegeProtection_t   PrivilegeProtection,
         MOTO_FLASH_A_TextDataProtection_t    TextDataProtection
         )
```

10.4.9.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
MOTO_FLASH_A_BlockSelect_t	BlocksToProtect	Determines blocks selected for program/erase operations	Any enumeration of MOTO_FLASH_A_BlockSelect_t, or values formed by a bitwise-or of this enumeration.
MOTO_FLASH_A_ReadWriteProtection_t	ReadWriteProtection	Determines read-only or read-write mode	Any enumeration of MOTO_FLASH_A_ReadWriteProtection_t
MOTO_FLASH_A_PrivilegeProtection_t	PrivilegeProtection	Determines supervisor-user or supervisor-only mode	Any enumeration of MOTO_FLASH_A_PrivilegeProtection_t
MOTO_FLASH_A_TextDataProtection_t	TextDataProtection	Determines text-only or text/data mode	Any enumeration of MOTO_FLASH_A_TextDataProtection_t

10.4.9.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
MOTO_FLASH_A_ERR_INVALID_BLOCK_CONFIG	Specified CMFR Flash Block is invalid
MOTO_FLASH_A_ERR_INVALID_PROTECTION	Specified Protection mode is invalid

10.4.10 MOTO_FLASH_A_SetPulseWidth

10.4.10.1 Description

MOTO_FLASH_A_SetPulseWidth configures the CLKPM, CLKPE, and SCLKR fields of the CMFRCTL register in order to set the specified pulse width of the specified program/erase cycle for the specified value of the system clock. Parameter checking is a compile-time option.

10.4.10.2 Prototype

```
ddErr_t MOTO_FLASH_A_SetPulseWidth (
         pMOTO_FLASH_A_t             FLASHPtr,
         MOTO_FLASH_A_PulseWidth_t   PulseWidth,
         MOTO_FLASH_A_SystemClock_t SystemClock,
         MOTO_FLASH_A_CycleType_t    CycleType
         )
```

10.4.10.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
MOTO_FLASH_A_PulseWidth_t	PulseWidth	Desired pulse width in units of tenths of microseconds	If CycleType = MOTO_FLASH_A_PROGRAM: 21.2uS to 32uS If CycleType = MOTO_FLASH_A_ERASE: 0.9S to 1.1S
MOTO_FLASH_A_SystemClock_t	SystemClock	Current value of system clock in units of Hertz	8 to 40 MHZ
MOTO_FLASH_A_CycleType_t	CycleType	Determines program/erase cycle type	MOTO_FLASH_A_PROGRAM or MOTO_FLASH_A_ERASE

10.4.10.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
MOTO_FLASH_A_ERR_INVALID_PULSE_WIDTH	Specified CMFR Flash pulse width is invalid
MOTO_FLASH_A_ERR_INVALID_SYSTEM_CLOCK	Specified system clock value is out of range
MOTO_FLASH_A_ERR_PULSE_ACTIVE	The program/erase pulse is active
MOTO_FLASH_A_ERR_INVALID_CYCLE_TYPE	The cycle type specified is not valid

10.4.11 MOTO_FLASH_A_GetPulseStatus

10.4.11.1 Description

MOTO_FLASH_A_GetPulseStatus returns a value indicating whether a programming or erase pulse is active. Parameter checking is a compile-time option.

10.4.11.2 Prototype

```
ddErr_t MOTO_FLASH_A_GetPulseStatus (
         pMOTO_FLASH_A_t            FLASHPtr,
         ddBoolean_t                *PulseStatus
         )
```

10.4.11.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
ddBoolean_t *	PulseStatus	Determines if a high voltage pulse is active	ddTRUE= High voltage pulse is active ddFALSE = High voltage pulse is not active

10.4.11.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
DD_ERR_BAD_RESULT_ADDRESS	PulseStatus pointer is zero

10.4.12 MOTO_FLASH_A_SelectBlocks

10.4.12.1 Description

MOTO_FLASH_A _SelectBlocks selects single or multiple blocks for erasing or programming, depending on user input. Parameter checking is a compile-time option.

10.4.12.2 Prototype

```
ddErr_t MOTO_FLASH_A_SelectBlocks (
         pMOTO_FLASH_A_t            FLASHPtr,
         MOTO_FLASH_A_BlockSelect_t          BlockSelect
         )
```

10.4.12.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
MOTO_FLASH_A_BlockSelect_t	BlockSelect	Determines blocks selected for program/erase operations	Any enumeration of MOTO_FLASH_A_BlockSelect_t, or values formed by a bitwise-or of this enumeration.

10.4.12.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
MOTO_FLASH_A_ERR_INVALID_BLOCK_CONFIG	Invalid Block configuration specified

10.4.13 MOTO_FLASH_A_GetHighVoltageEnableStatus

10.4.13.1 Description

MOTO_FLASH_A_GetHighVoltageEnableStatus determines whether high voltage operations necessary for programming or erasing are possible. Parameter checking is a compile-time option.

10.4.13.2 Prototype

```
DdErr_t MOTO_FLASH_A_GetHighVoltageEnableStatus (
                    pMOTO_FLASH_A_t       FLASHPtr,
                    ddBoolean_t           *HighVoltageEnableStatus
         )
```

10.4.13.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
DdBoolean_t *	HighVoltageEnableStatus	Determines if high voltage operations are enabled	ddTRUE= High voltage operations are enabled ddFALSE = High voltage operations are disabled

10.4.13.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
DD_ERR_BAD_RESULT_ADDRESS	HighVoltageEnableStatus pointer is zero

10.4.14 MOTO_FLASH_A_ControlArrayWriteMode

10.4.14.1 Description

MOTO_FLASH_A _ControlArrayWriteMode configures the array write mode to program or erase, depending on user input. Parameter checking is a compile-time option.

10.4.14.2 Prototype

```
DdErr_t MOTO_FLASH_A_ControlArrayWriteMode (
         pMOTO_FLASH_A_t                    FLASHPtr,
         MOTO_FLASH_A_CycleType_t           WriteMode
      )
```

10.4.14.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
MOTO_FLASH_A_CycleType_t	WriteMode	Determines array access mode of operation	MOTO_FLASH_A_PROGRAM = write mode is program MOTO_FLASH_A_ERASE = write mode is erase

10.4.14.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
MOTO_FLASH_A_ERR_INVALID_CYCLE_TYPE	Cycle type specified is invalid

10.4.15 MOTO_FLASH_A_ControlSequence

10.4.15.1 Description

MOTO_FLASH_A _ControlSequence starts or ends a program/erase sequence, depending on user input. Parameter checking is a compile-time option.

10.4.15.2 Prototype

```
ddErr_t MOTO_FLASH_A_ControlSequence (
         pMOTO_FLASH_A_t            FLASHPtr,
         MOTO_FLASH_A_Sequence_t    ModeControl
      )
```

10.4.15.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
MOTO_FLASH_A_Sequence_t	ModeControl	Determines start or end program sequence mode of operation	MOTO_FLASH_A_START= start a program/erase sequence MOTO_FLASH_A_END = end a program/erase sequence

10.4.15.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
MOTO_FLASH_A_ERR_INVALID_MODE	Mode control input is invalid

10.4.16 MOTO_FLASH_A_ControlPulse

10.4.16.1 Description

MOTO_FLASH_A _ControlPulse enables or disables a program/erase pulse, depending on user input. Parameter checking is a compile-time option.

10.4.16.2 Prototype

```
ddErr_t MOTO_FLASH_A_ControlPulse (
         pMOTO_FLASH_A_t            FLASHPtr,
         MOTO_FLASH_A_Control_t     ModeControl
      )
```

10.4.16.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
MOTO_FLASH_A_Control_t	ModeControl	Determines program/erase pulse mode of operation	MOTO_FLASH_A_ENABLE= enable a program/erase pulse MOTO_FLASH_A_DISABLE = disable a program/erase pulse

10.4.16.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
MOTO_FLASH_A_ERR_INVALID_MODE	Mode control input is invalid

10.4.17 MOTO_FLASH_A_GetRegister

10.4.17.1 Description

MOTO_FLASH_A_GetRegister returns the contents of the specified CMFR Flash register. Parameter checking is a compile-time option.

10.4.17.2 Prototype

```
ddErr_t MOTO_FLASH_A_GetRegister (
         pMOTO_FLASH_A_t                 CMFR FlashPtr,
         MOTO_FLASH_A_RegisterSwitch_t   RegSwitch,
         UINT32                          *GetRegisterPtr
         )
```

10.4.17.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	CMFR FlashPtr	MOTO_FLASH_A base address associated with this driver.	Any non-zero core processor data address.
MOTO_FLASH_A_RegisterSwitch_t	RegSwitch	Specifies which register to read	Any enumerated value of MOTO_FLASH_A_RegisterSwitch_t
UINT32 *	GetRegisterPtr	Result address for selected MOTO_FLASH_A register.	Any non-zero core processor data address.

10.4.17.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	CMFR Flash base address parameter is zero
DD_ERR_BAD_RESULT_ADDR	Result Pointer is zero
DD_ERR_INVALID_REGISTER	MOTO_FLASH_A Register Switch is invalid

10.4.18 MOTO_FLASH_A_SetRegister

10.4.18.1 Description

MOTO_FLASH_A_SetRegister writes a 32-bit data value to the specified CMFR Flash register. Parameter checking is a compile-time option.

10.4.18.2 Prototype

```
ddErr_t MOTO_FLASH_A_SetRegister (
        pMOTO_FLASH_A_t                     CMFR FlashPtr,
        MOTO_FLASH_A_RegisterSwitch_t       RegSwitch,
        UINT32                              RegisterValue
)
```

10.4.18.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	CMFR FlashPtr	MOTO_FLASH_A base address associated with this driver.	Any non-zero core processor data address.
MOTO_FLASH_A_RegisterSwitch_t	RegSwitch	Specifies which register to read	Any enumerated value of MOTO_FLASH_A_RegisterSwitch_t
UINT32	RegisterValue	Data to copy into selected MOTO_FLASH_A register.	Any 32-bit value.

10.4.18.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	CMFR Flash base address parameter is zero
DD_ERR_INVALID_REGISTER	MOTO_FLASH_A Register Switch is invalid

10.4.19 MOTO_FLASH_A_EraseSerialMRVerify

10.4.19.1 Description

MOTO_FLASH_A_EraseSerialMRVerify erases multiple blocks within a single flash array or its shadow region based on user input. Parameter checking is a compile-time option.

10.4.19.2 Prototype

```
ddErr_t MOTO_FLASH_A_EraseSerialMRVerify(
         pMOTO_FLASH_A_t                    Moto_FLASHPtr,
         pMOTO_FLASH_A_Address_t            DeviceBaseAddress,
         MOTO_FLASH_A_BlockSelect_t         BlocksToErase,
         MOTO_FLASH_A_DataWidth_t           DataWidth,
         MOTO_FLASH_A_BlockSize_t           BlockSize,
         MOTO_FLASH_A_ShadowBlock_t         ShadowBlock,
         MOTO_FLASH_A_ShadowStartOffset_t   ShadowStartOffset,
         MOTO_FLASH_A_ShadowEndOffset_t     ShadowEndOffset,
         MOTO_FLASH_A_MaxPulses_t           MaxPulses
      )
```

10.4.19.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	Moto_FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
pMOTO_FLASH_A_Address_t	DeviceBaseAddress	MOTO_FLASH_A array base address associated with this driver.	Any core processor data address.

MOTO_FLASH_A_BlockSelect_t	BlocksToErase	Determines which block or blocks are selected for write operations	Any enumerated value of MOTO_FLASH_A_BlockSelect_t, or bitwise or of its enumerated values
MOTO_FLASH_A_DataWidth_t	DataWidth	Determines memory width in units of bytes	Any enumerated value of MOTO_FLASH_A_DataWidth_t
MOTO_FLASH_A_BlockSize_t	BlockSize	Determines array block size	Any enumerated value of MOTO_FLASH_A_BlockSize_t
MOTO_FLASH_A_ShadowBlock_t	ShadowBlock	Determines shadow block number	Any enumerated value of MOTO_FLASH_A_ShadowBlock_t
MOTO_FLASH_A_ShadowStartOffset_t	ShadowStartOffset	Determines start of shadow information as an offset of bytes from device base	Any enumerated value of MOTO_FLASH_A_ShadowStartOffset_t
MOTO_FLASH_A_ShadowEndOffset_t	ShadowEndOffset	Determines end of shadow information as an offset of bytes from device base	Any enumerated value of MOTO_FLASH_A_ShadowEndOffset_t
MOTO_FLASH_A_MaxPulses_t	MaxPulses	Maximum number of Erase Pulses	1

10.4.19.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
DD_ERR_INVALID_ADDRESS	Specified address pointer is zero
MOTO_FLASH_A_ERR_INVALID_BLOCK_CONFIG	Specified block config value is out of range
MOTO_FLASH_A_ERR_INVALID_DATA_WIDTH	Specified data width value is out of range
MOTO_FLASH_A_ERR_INVALID_BLOCK_SIZE	Specified block size value is out of range
MOTO_FLASH_A_ERR_INVALID_SHADOW_BLOCK	Specified shadow block value is out of range
MOTO_FLASH_A_ERR_INVALID_SHADOW_OFFSET	Specified shadow offset value is out of range
MOTO_FLASH_A_ERR_INVALID_MAX_PULSES	Specified max pulses value is out of range
MOTO_FLASH_A_ERR_ARRAY_ERASE_FAILURE	Failure to erase the array within specified number of max pulses
MOTO_FLASH_A_ERR_SHADOW_ERASE_FAILURE	Failure to erase the shadow information within specified number of max pulses

10.4.20 MOTO_FLASH_A_EraseParallelMRVerify

10.4.20.1 Description

MOTO_FLASH_A_EraseParallelMRVerify erases multiple blocks within multiple flash arrays and their shadow regions based on user input. Parameter checking is a compile-time option.

10.4.20.2 Prototype

```
ddErr_t MOTO_FLASH_A_EraseParallelMRVerify(
         pMOTO_FLASH_A_t                    Moto_FLASHPtr[],
         pMOTO_FLASH_A_Address_t            DeviceBaseAddress[],
         UINT32                             DeviceCount,
         MOTO_FLASH_A_BlockSelect_t         BlocksToErase,
         MOTO_FLASH_A_DataWidth_t           DataWidth,
         MOTO_FLASH_A_BlockSize_t           BlockSize,
         MOTO_FLASH_A_ShadowBlock_t         ShadowBlock,
         MOTO_FLASH_A_ShadowStartOffset_t   ShadowStartOffset,
         MOTO_FLASH_A_ShadowEndOffset_t     ShadowEndOffset,
         MOTO_FLASH_A_MaxPulses_t           MaxPulses
      )
```

10.4.20.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	Moto_FLASHPtr	Array of MOTO_FLASH_A register base addresses associated with this driver.	Any non-zero core processor data address.
pMOTO_FLASH_A_Address_t	DeviceBaseAddress	Array of MOTO_FLASH_A array base addresses associated with this driver.	Any core processor data address.
UINT32	DeviceCount	Number of flash device modules to erase in a parallel manner	Any value from one to MOTO_FLASH_A_MAX_NUMBER_OF_DEVICES
MOTO_FLASH_A_BlockSelect_t	BlocksToErase	Determines which block or blocks are selected for write operations	Any enumerated value of MOTO_FLASH_A_BlockSelect_t, or bitwise or of its enumerated values
MOTO_FLASH_A_DataWidth_t	DataWidth	Determines memory width in units of bytes	Any enumerated value of MOTO_FLASH_A_DataWidth_t
MOTO_FLASH_A_BlockSize_t	BlockSize	Determines array block size	Any enumerated value of MOTO_FLASH_A_BlockSize_t
MOTO_FLASH_A_ShadowBlock_t	ShadowBlock	Determines shadow block number	Any enumerated value of MOTO_FLASH_A_ShadowBlock_t
MOTO_FLASH_A_ShadowStartOffset_t	ShadowStartOffset	Determines start of shadow information as an offset of bytes from device base	Any enumerated value of MOTO_FLASH_A_ShadowStartOffset_t
MOTO_FLASH_A_ShadowEndOffset_t	ShadowEndOffset	Determines end of shadow information as an offset of bytes from device base	Any enumerated value of MOTO_FLASH_A_ShadowEndOffset_t

MOTO_FLASH_A_SystemClock_t	SystemClock	System Clock value in units of HZ	If CycleType = MOTO_FLASH_A_PROGRAM or MOTO_FLASH_A_ERASE: 8 to 40 MHZ if CycleType = MOTO_FLASH_A_READ: NULL
MOTO_FLASH_A_PulseWidth_t	PulseWidth	Desired pulse width in units of microseconds	If CycleType = MOTO_FLASH_A_PROGRAM: 21.2uS to 32uS If CycleType = MOTO_FLASH_A_ERASE: 0.9S to 1.1S if CycleType = READ: NULL
MOTO_FLASH_A_MaxPulses_t	MaxPulses	Maximum number of Erase Pulses	1

10.4.20.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
DD_ERR_INVALID_ADDRESS	Specified address pointer is zero
MOTO_FLASH_A_ERR_INVALID_BLOCK_CONFIG	Specified block config value is out of range
MOTO_FLASH_A_ERR_INVALID_DATA_WIDTH	Specified data width value is out of range
MOTO_FLASH_A_ERR_INVALID_BLOCK_SIZE	Specified block size value is out of range
MOTO_FLASH_A_ERR_INVALID_SHADOW_BLOCK	Specified shadow block value is out of range
MOTO_FLASH_A_ERR_INVALID_SHADOW_OFFSET	Specified shadow offset value is out of range
MOTO_FLASH_A_ERR_INVALID_MAX_PULSES	Specified max pulses value is out of range
MOTO_FLASH_A_ERR_INVALID_DEVICE_COUNT	Specified device count is out of range
MOTO_FLASH_A_ERR_ARRAY_ERASE_FAILURE	Failure to erase the array within specified number of max pulses
MOTO_FLASH_A_ERR_SHADOW_ERASE_FAILURE	Failure to erase shadow information within specified number of max pulses

10.4.21 MOTO_FLASH_A_ProgramSerialMRVerify

10.4.21.1 Description

MOTO_FLASH_A_ProgramSerialMRVerify programs multiple blocks within a single flash array or its shadow region based on user input. Parameter checking is a compile-time option.

10.4.21.2 Prototype

```
ddErr_t MOTO_FLASH_A_ProgramSerialMRVerify(
         pMOTO_FLASH_A_t                    Moto_FLASHPtr,
         pMOTO_FLASH_A_Address_t            DeviceBaseAddress,
         pMOTO_FLASH_A_Address_t            SourceBuffer,
```

MOTO_FLASH_A Level 1

```
        MOTO_FLASH_A_BlockSelect_t          StartBlock,
        MOTO_FLASH_A_Control_t              ShadowAccess,
        UINT32                              DataLength,
        MOTO_FLASH_A_DataWidth_t            DataWidth,
        MOTO_FLASH_A_BlockSize_t            BlockSize,
        MOTO_FLASH_A_MaxPulses_t            MaxPulses
    )
```

10.4.21.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_t	Moto_FLASHPtr	MOTO_FLASH_A register base address associated with this driver.	Any non-zero core processor data address.
pMOTO_FLASH_A_Address_t	DeviceBaseAddress	MOTO_FLASH_A array base address associated with this driver.	Any core processor data address.
pMOTO_FLASH_A_Address_t	SourceBuffer	Points to the beginning of data to program.	Any non-zero core processor data address.
MOTO_FLASH_A_BlockSelect_t	StartBlock	Determines which block will be assigned as the beginning block	Any enumerated value of MOTO_FLASH_A_BlockSelect_t, or bitwise or of its enumerated values
MOTO_FLASH_A_Control_t	ShadowAccess	Specifies shadow or array programming	Any enumerated value of MOTO_FLASH_A_Control_t
UINT32	DataLength	Specifies length of data to program in bytes	Any 32-bit integer value less than or equal to the number of bytes in the device
MOTO_FLASH_A_DataWidth_t	DataWidth	Determines memory width in units of bytes	Any enumerated value of MOTO_FLASH_A_DataWidth_t
MOTO_FLASH_A_BlockSize_t	BlockSize	Determines array block size	Any enumerated value of MOTO_FLASH_A_BlockSize_t
MOTO_FLASH_A_MaxPulses_t	MaxPulses	Maximum number of Erase Pulses	1

10.4.21.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_HANDLE	MOTO FLASH base address parameter is zero
DD_ERR_INVALID_ADDRESS	Specified address pointer is zero
MOTO_FLASH_A_ERR_INVALID_BLOCK_CONFIG	Specified block config value is out of range
MOTO_FLASH_A_ERR_INVALID_DATA_WIDTH	Specified data width value is out of range
MOTO_FLASH_A_ERR_INVALID_BLOCK_SIZE	Specified block size value is out of range
MOTO_FLASH_A_ERR_INVALID_MAX_PULSES	Specified max pulses value is out of range
MOTO_FLASH_A_ERR_PROGRAM_FAILURE	Failure to program array within specified max number of pulses

10.4.22 MOTO_FLASH_A_BlankCheck

10.4.22.1 Description

MOTO_FLASH_A_BlankCheck checks the array or shadow information for blank content, and returns an error if not blank. Parameter checking is a compile-time option.

10.4.22.2 Prototype

```
ddErr_t MOTO_FLASH_A_BlankCheck(

        pMOTO_FLASH_A_Address_t          StartAddress,
        pMOTO_FLASH_A_Address_t          EndAddress,
        MOTO_FLASH_A_Data_t              BlankValue,
        MOTO_FLASH_A_DataWidth_t         DataWidth,
    )
```

10.4.22.3 Arguments

Data Type	Formal Name	Description	Valid Values
pMOTO_FLASH_A_Address_t	StartAddress	Starting address to blank check.	Any core processor data address.
pMOTO_FLASH_A_Address_t	EndAddress	Ending address to blank check.	Any non-zero core processor data address.
MOTO_FLASH_A_Data_t	BlankValue	The value corresponding to an erased bitcell state	Any value from zero to the max that MOTO_FLASH_A_Data_t can store.
MOTO_FLASH_A_DataWidth_t	DataWidth	Determines memory width in units of bytes	Any enumerated value of MOTO_FLASH_A_DataWidth_t

10.4.22.4 Return Values

Error Code	Description
DD_ERR_NONE	No error
DD_ERR_INVALID_ADDRESS	Specified address pointer is zero
MOTO_FLASH_A_ERR_INVALID_DATA_WIDTH	Specified data width value is out of range
MOTO_FLASH_A_ERR_DEVICE_NOT_BLANK	Device is not blank in the address range specified

10.5 MOTO_FLASH_A Example Application

The following describes the MOTO_FLASH_A example application.

10.5.1 Description for Example Application

The MOTO_FLASH_A example program illustrates use of the MOTO_FLASH_A driver. It operates on an M•CORE MMC2107 board. The program demonstrates how to do an erase, program, and read of the flash.

10.5.1.1 Example Application Code

```
/*********************************************************************
                Example Application for MOTO_FLASH_A Module

File: motoflasha_app.c

Purpose: This program executes an example application of the M-CORE
Peripherals Library MOTO_FLASH_A Level 1 driver.  It shows how to erase
and program a block of memory.

If an error occurs, the program will return the line number of the
failing test.

Time to Run: Approximately one second.

General Setup:
          General hardware and software setup to run this application
          can be found in the 'readme_app.txt' file, located in the
          '<library>/app' directory.

Application Specific Setup:
          Development Board: MMC2107.

(C) Copyright Motorola Inc, 2000.  All rights reserved.
*********************************************************************/

#include "plibdefs_mmc2107.h"
#include "moto_flash_a.h"

/*********************************************************************/
/* Macro Definitions                                                 */
/*********************************************************************/

#define HANDLE_ERROR(err) if(err != MOTO_FLASH_A_ERR_NONE)            \
                          return(__LINE__)

#define MMIO_ENABLEVPP_BITNO 5
#define MMIO_ENABLEVPP_MASK (1<<MMIO_ENABLEVPP_BITNO)

/* Flash addresses for read operations - (blank check and verify)  */
```

```
#define EXAMPLE_APP_BLOCK 0

/*********************************************************************/
/* Function: main                                                    */
/*                                                                   */
/* Purpose: Executes MOTO_FLASH_A modules for MOTO_FLASH_A Level 1   */
/*          example app.                                             */
/*                                                                   */
/* Inputs:  none                                                     */
/*                                                                   */
/* Outputs: returns line number of error condition, zero if sucessful */
/*********************************************************************/
int main (void)
{
    int status;

    status = motoflasha_example();

    return(status);

}


/*********************************************************************/
/* Function: motoflasha_example                                      */
/*                                                                   */
/* Purpose: Executes MOTO_FLASH_A modules for MOTO_FLASH_A Level 1   */
/*          example app.                                             */
/*                                                                   */
/* Inputs:  none                                                     */
/*                                                                   */
/* Outputs: returns line number of error condition, zero if sucessful */
/*********************************************************************/
int motoflasha_example (void)
{
   MOTO_FLASH_A_BlockSelect_t ProtectedBlocks;   /* Blocks to protect */
   MOTO_FLASH_A_BlockSelect_t StartBlock;   /* Block to erase/program */
   pMOTO_FLASH_A_Address_t StartAddress;          /* Starting address */
   pMOTO_FLASH_A_Address_t EndAddress;              /* Ending address */
   MOTO_FLASH_A_Data_t BlankValue;          /* Value of erased memory */
   MOTO_FLASH_A_Control_t ShadowAccess;           /* Access to shadow */
   UINT32 Length;                                   /* Length of data */
   UINT32 Buffer[ (MOTO_FLASH_A_16K_BYTES /
                   sizeof(UINT32)) ];                  /*1 blk. patt*/
   UINT32 wordcnt;                                  /* loop counter */

   UINT8* pMMIO = (UINT8*)__MMC2107_MMIO;        /* MMIO base address */
   pMOTO_FLASH_A_t pCMFR = (pMOTO_FLASH_A_t)__MMC2107_CMFR_CTL;/*FLASH*/

   MOTO_FLASH_A_ReturnCode_t retval;                 /* return value */


   StartAddress = (UINT32)(__MMC2107_CMFR_FLASH+
                           ( EXAMPLE_APP_BLOCK *MOTO_FLASH_A_16K_BYTES)
```

```
                        );
    EndAddress = (UINT32)(StartAddress+MOTO_FLASH_A_16K_BYTES);

    BlankValue = MOTO_FLASH_A_BLANK_VALUE;


    /* Enable MMCCMB2103 VPP (bit 5 of MMIO) */
    *pMMIO = MMIO_ENABLEVPP_MASK;

    /* Select blocks for protection */
    ProtectedBlocks = MOTO_FLASH_A_BLOCK_0;

    /* Initialize for Erasing */
    retval = MOTO_FLASH_A_Init(
                          pCMFR,                /* FLASH base address */
                          MOTO_FLASH_A_ERASE,          /* Cycle type */
                          MOTO_FLASH_A_BLOCK_0,       /* Select block */
                          MOTO_FLASH_A_DISABLE,      /* Enable shadow */
                          (MOTO_FLASH_A_SystemClock_t) 32,  /* 32 MHZ */
                          (MOTO_FLASH_A_PulseWidth_t) 1100000,/*1.1 s */
                          ProtectedBlocks,       /* blocks to protect */
                          MOTO_FLASH_A_READ_WRITE,/*Allow reads/writes*/
                          MOTO_FLASH_A_USER_ACCESS,/*Allow user access*/
                          MOTO_FLASH_A_TEXT_AND_DATA_ACCESS);

    HANDLE_ERROR(retval);

    /* Erase block 0 of CMFR FLASH */
    retval = MOTO_FLASH_A_EraseSerialMRVerify(
                         pCMFR, /* FLASH base address */
                         ((pMOTO_FLASH_A_Address_t)__MMC2107_CMFR_FLASH),
                         (MOTO_FLASH_A_BLOCK_0),   /* Block to erase */
                         MOTO_FLASH_A_4_BYTES,         /* Data width */
                         MOTO_FLASH_A_16K_BYTES,       /* Block size */
                         MOTO_FLASH_A_LOW_FILL_SHADOW_BLOCK,/* Shadow*/
                         MOTO_FLASH_A_SHADOW_START_OFFSET_0_BYTES,
                         MOTO_FLASH_A_SHADOW_END_OFFSET_256_BYTES,
                         MOTO_FLASH_A_MAX_NUMBER_ERASE_PULSES);

    HANDLE_ERROR(retval);

    /* Verify memory was erased */
    retval = MOTO_FLASH_A_BlankCheck(
                           StartAddress,/*Start address to check*/
                          EndAddress,  /* End address to check */
                        MOTO_FLASH_A_4_BYTES   /* Data width */
                            );
    HANDLE_ERROR(retval);

    /* Initialize for programming */
    retval = MOTO_FLASH_A_Init(
                          pCMFR,                /* FLASH base address */
                          MOTO_FLASH_A_PROGRAM,         /* Cycle type */
                          MOTO_FLASH_A_BLOCK_0,       /* Select block */
```

```
                        MOTO_FLASH_A_DISABLE,      /* Enable shadow */
                        (MOTO_FLASH_A_SystemClock_t) 32,  /* 32 MHZ */
                        (MOTO_FLASH_A_PulseWidth_t) 16,   /* 16 usec*/
                        ProtectedBlocks,       /* Blocks to protect */
                        MOTO_FLASH_A_READ_WRITE,/*Allow reads/writes*/
                        MOTO_FLASH_A_USER_ACCESS,/*Allow user access*/
                        MOTO_FLASH_A_TEXT_AND_DATA_ACCESS);

   HANDLE_ERROR(retval);

   /* Program block 0 of CMFR FLASH */
   StartBlock = MOTO_FLASH_A_BLOCK_0;   /* Block of starting address */
   ShadowAccess = MOTO_FLASH_A_DISABLE;     /* Disable shadow access */
   Length = MOTO_FLASH_A_16K_BYTES;     /* Length of data to program */

   for( wordcnt = 0;
        wordcnt < (MOTO_FLASH_A_16K_BYTES/sizeof(Buffer[0]));
        wordcnt++ )
   {
       Buffer[wordcnt]=wordcnt;  /* init data buffer for programming */
   } /* end for loop */

   retval = MOTO_FLASH_A_ProgramSerialMRVerify(
                       pCMFR,/* FLASH base address*/
                       ((pMOTO_FLASH_A_Address_t)__MMC2107_CMFR_FLASH),
                       (pMOTO_FLASH_A_Address_t)Buffer,/* Data buffer */
                       StartBlock,                 /* Beginning block */
                       ShadowAccess,                 /* Shadow access */
                       Length,                        /* Data length */
                       MOTO_FLASH_A_4_BYTES,           /* Data width */
                       MOTO_FLASH_A_16K_BYTES,         /* Block size */
                       MOTO_FLASH_A_MAX_NUMBER_PROGRAM_PULSES);

   HANDLE_ERROR(retval);

  /* Verify Flash data matches the program data buffer */
   for( wordcnt = 0;
        wordcnt < (MOTO_FLASH_A_16K_BYTES/sizeof(Buffer[0]));
        wordcnt++, StartAddress += sizeof(StartAddress))
   {
      if ( (*(UINT32 *)StartAddress) != Buffer[wordcnt] )
      {
        return(__LINE__);  /* return error if mismatch found */
      } /* end if */
   } /* end for loop */

return(MOTO_FLASH_A_ERR_NONE);

} /* end of motoflasha_example */
```

Section 11 PIT_B Level 1

This section describes the PIT_B Level 1 M•CORE™ Device Driver.

11.1 Overview

The Programable Interval Timer (PIT) and its accompanying driver (PIT_B) are used on the MMC2107 development system. The level 1 service is designed to allow the programmer to control the PIT on a near-register level, but without the concern of exact positions of registers and fields, or the detailed sequences of register operations. Refer to the MMC2107 Advance Information (Data Book) (MMC2107/D) for additional information.

11.2 PIT_B Service Functions

The PIT_B device driver software performs these functions:

- Initializes the Programable Interval Timer (PIT).
- Enables and disables the PIT Interrupt Enable (PIE).
- Gets the PIT interrupt status.
- Enables and disables the PIT.
- Loads a 16-bit data value into the modulus.
- Writes a 16-bit data value to a sub-set of PIT registers.
- Gets the contents of a sub-set of PIT registers.
- Resets the PIT.

11.3 PIT_B Data Type Definitions

The following paragraphs describe the data definitions recognized by the PIT_B device driver. These data definitions are grouped into six categories: standard data types, board specific base address definitions, abstract data types, enumerated data types, register macros, and return codes.

11.3.1 Standard Data Type Definitions

- UINT8 — 8-bit unsigned data value
- INT8 — 8-bit signed data value
- UINT16 — 16-bit unsigned data value
- INT16 — 16-bit signed data value

11.3.2 Board Specific Base Address Definitions

```
#define __MMC2107_PIT0   (0x00C8_0000)
#define __MMC2107_PIT1   (0x00C9_0000)
```

11.3.3 Abstract Data Type Definitions

11.3.3.1 PIT_B_t - PIT Register Definition

```
typedef struct {
   volatile UINT16   PCSR; /* Programmable Interval Timer Control/Status Register*/
            UINT16   PMR;  /* Programmable Interval Timer Modulus Register       */
   volatile UINT16   PCNTR; /* Programmable Interval Timer Count Register        */
} PIT_B_t, *pPIT_B_t;
```

11.3.4 Enumerated Data Type Definitions

11.3.4.1 PIT_B_Register_t - Programmable Interval Timer Register Selection

```
typedef enum {
    PIT_B_PCSR,   /* Select PIT Control/Status Register */
    PIT_B_PMR,    /* Select PIT Modulus Register        */
    PIT_B_PCNTR   /* Select PIT Count Register          */
} PIT_B_Register_t;
```

11.3.4.2 PIT_B_PITEnable_t - PIT Enable

```
typedef enum {
    PIT_B_FUNCTION_DISABLE,       /* Disables the PIT    */
    PIT_B_FUNCTION_ENABLE,        /* Enables the PIT     */
} PIT_B_PITEnable_t;
```

11.3.4.3 PIT_B_CounterReloadControl_t - Counter Reload Control Selection

```
typedef enum {
    PIT_B_COUNTER_RELOAD_ROLLS_OVER,   /* Rolls over to 0xFFFF             */
    PIT_B_COUNTER_RELOAD_FROM_MODULUS  /* Reloaded from the modulus latch  */
} PIT_B_CounterReloadControl_t;
```

11.3.4.4 PIT_B_PITInterruptFlag_t - PIT Interrupt Flag Selection

```
typedef enum {
    PIT_B_NO_INTERRUPT_PRESENT,     /* No PIT interrupt is present  */
    PIT_B_INTERRUPT_PRESENT         /* PIT interrupt is present     */
} PIT_B_PITInterruptFlag_t;
```

11.3.4.5 PIT_B_PITInterruptEnable_t - PIT Interrupt Enable Selection

```
typedef enum {
    PIT_B_INTERRUPT_ENABLE, /* Interrupt flag is inhibited from reaching the CPU */
    PIT_B_INTERRUPT_DISABLE /* Interrupt flag is allowed to request an interrupt */
} PIT_B_PITInterruptEnable_t;
```

11.3.4.6 PIT_B_CounterOverwriteEnable_t - Counter Overwrite Enable Selection

```
typedef enum {
    PIT_B_DISABLE_COUNTER_OVERWRITE,    /* Modulus latch holds register */
    PIT_B_ENABLE_COUNTER_OVERWRITE      /* Modulus latch is transparent */
} PIT_B_CounterOverwriteEnable_t;
```

11.3.4.7 PIT_B_DEBUGModeControl_t -DEBUG Mode Control Selection

```
typedef enum {
    ENABLE_PIT_B_IN_DEBUG,   /* PIT function is not affected in DEBUG mode */
    DISABLE_PIT_B_IN_DEBUG   /* PIT function is stopped in DEBUG mode      */
} PIT_B_DEBUGModeControl_t;
```

11.3.4.8 PIT_B_DOZEModeControl_t - DOZE Mode Control Selection

```
typedef enum {
    ENABLE_PIT_B_IN_DOZE,   /* PIT function is not affected in DOZE mode */
    DISABLE_PIT_B_IN_DOZE   /* PIT function is stopped in DOZE mode      */
} PIT_B_DOZEModeControl_t;
```

11.3.4.9 PIT_B_PreScalarSelect_t - Pre-Scalar Select Encoding

```
typedef enum {
 PIT_B_SCALAR_00, /*Pre-Scaler select value is equal to the system clock         */
 PIT_B_SCALAR_01, /*Pre-Scaler select value is equal to the (system clock)/2     */
 PIT_B_SCALAR_02, /*Pre-Scaler select value is equal to the (system clock)/4     */
 PIT_B_SCALAR_03, /*Pre-Scaler select value is equal to the (system clock)/8     */
 PIT_B_SCALAR_04, /*Pre-Scaler select value is equal to the (system clock)/16    */
 PIT_B_SCALAR_05, /*Pre-Scaler select value is equal to the (system clock)/32    */
 PIT_B_SCALAR_06, /*Pre-Scaler select value is equal to the (system clock)/64    */
 PIT_B_SCALAR_07, /*Pre-Scaler select value is equal to the (system clock)/128   */
 PIT_B_SCALAR_08, /*Pre-Scaler select value is equal to the (system clock)/256   */
 PIT_B_SCALAR_09, /*Pre-Scaler select value is equal to the (system clock)/512   */
 PIT_B_SCALAR_10, /*Pre-Scaler select value is equal to the (system clock)/1024  */
 PIT_B_SCALAR_11, /*Pre-Scaler select value is equal to the (system clock)/2048  */
 PIT_B_SCALAR_12, /*Pre-Scaler select value is equal to the (system clock)/4096  */
 PIT_B_SCALAR_13, /*Pre-Scaler select value is equal to the (system clock)/8192  */
 PIT_B_SCALAR_14, /*Pre-Scaler select value is equal to the (system clock)/16384 */
 PIT_B_SCALAR_15, /*Pre-Scaler select value is equal to the (system clock)/32768 */
} PIT_B_PreScalarSelect_t;
```

11.3.4.10 PIT_B_ReturnCode_t - Error Return Code Selection

```
typedef enum{
    PIT_B_ERR_NONE,
    PIT_B_ERR_INVALID_HANDLE,
    PIT_B_ERR_INVALID_COUNTER_RELOAD_CONTROL,
    PIT_B_ERR_INVALID_COUNTER_OVERWRITE_ENABLE,
    PIT_B_ERR_INVALID_DEBUG_MODE_CONTROL,
    PIT_B_ERR_INVALID_DOZE_MODE_CONTROL,
    PIT_B_ERR_INVALID_PIT_INTERRUPT_ENABLE,
    PIT_B_ERR_INVALID_PRE_SCALAR_SELECT,
    PIT_B_ERR_INVALID_PIT_ENABLE,
    PIT_B_ERR_BAD_RESULT_ADDRESS,
    PIT_B_ERR_INVALID_PIT_REGISTER
} PIT_B_ReturnCode_t;        /*Pit_B return codes     */
```

11.3.5 Register Macros

11.3.5.1 PIT Control/Status Register (PCSR)

```
#define PCSR_EN_BITNO 0
#define PCSR_EN_MASK   (0 << PCSR_EN_BITNO)
#define PCSR_RLD_BITNO 1
#define PCSR_RLD_MASK  (1 << PCSR_RLD_BITNO)
#define PCSR_PIF_BITNO 2
#define PCSR_PIF_MASK  (1 << PCSR_PITIF_BITNO)
#define PCSR_PIE_BITNO 3
#define PCSR_PIE_MASK  (1 << PCSR_PITIE_BITNO)
#define PCSR_OVW_BITNO 4
#define PCSR_OVW_MASK  (1 << PCSR_OVW_BITNO)
#define PCSR_DBG_BITNO 5
#define PCSR_DBG_MASK  (1 << PCSR_DBG_BITNO)
#define PCSR_DOZE_BITNO 6
#define PCSR_DOZE_MASK (1 << PCSR_DOZE_BITNO)
#define PCSR_PRE_BITNO 8
#define PCSR_PRE_MAX 0xF
#define PCSR_PRE_MASK  (PCSR_PRE_MAX << PITCSR_PRE_BITNO)
#define PCSR_RESET_MASK  0x0
```

11.3.6 Return Codes

Return codes from the enumerated type “PIT_B_ReturnCode_t”.

Return Code	Functions Returning	Description
PIT_B_ERR_NONE	All	No error.
PIT_B_ERR_INVALID_HANDLE	All	PIT base address parameter is zero.
PIT_B_ERR_INVALID_COUNTER_RELOAD_CONTROL	PIT_B_Init, PIT_B_Reset	Counter reload control is invalid.
PIT_B_ERR_INVALID_COUNTER_OVERWRITE_ENABLE	PIT_B_Init	Counter overwrite enable is invalid.
PIT_B_ERR_INVALID_DEBUG_MODE_CONTROL	PIT_B_Init	DEBUG mode control is invalid.
PIT_B_ERR_INVALID_DOZE_MODE_CONTROL	PIT_B_Init	DOZE mode control is invalid.
PIT_B_ERR_INVALID_PIT_INTERRUPT_ENABLE	PIT_B_ControllInterrupt	PIT interrupt enable is invalid.
PIT_B_ERR_INVALID_PRE_SCALAR_SELECT	PIT_B_Init, PIT_B_ControllOperation	Pre-scalar select is invalid.
PIT_B_ERR_INVALID_PIT_ENABLE	PIT_B_ControllOperation	PIT enable is invalid.
PIT_B_ERR_BAD_RESULT_ADDRESS	PIT_B_GetRegister	Result Pointer is zero.
PIT_B_ERR_INVALID_PIT_REGISTER	PIT_B_GetRegister, PIT_B_SetRegister	PIT register is invalid.

11.4 PIT_B API Definitions

11.4.1 PIT_B_Init

11.4.1.1 Description

The PIT_B_Init function initializes the Programmable Interval Timer (PIT). Parameter checking is a compile-time option.

The PIT_B_Init function should be called once during the system software initialization.

11.4.1.2 Prototype

```
PIT_B_ReturnCode_t  PIT_B_Init (
          pPIT_B_t                          PITPtr,
          PIT_B_CounterReloadControl_t      CounterReloadControl,
          PIT_B_CounterOverwriteEnable_t    CounterOverwriteEnable,
          PIT_B_DEBUGModeControl_t          DEBUGModeControl,
          PIT_B_DOZEModeControl_t           DOZEModeControl,
          PIT_B_PreScalarSelect_t           PreScalarSelect,
          UINT16                            ModulusValue
)
```

11.4.1.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPIT_B_t	PITPtr	PIT base address associated with this driver	Any non-zero core processor data address.
PIT_B_CounterReloadControl_t	CounterReloadControl	Determines whether the PIT rolls over to a programmed interval or at 0xFFFF	PIT_B_COUNTER_RELOAD_ROLLS_OVER, PIT_B_COUNTER_RELOAD_FROM_MODULUS
PIT_B_CounterOverwriteEnable_t	CounterOverwriteEnable	Determines whether the modulus latch is transparent or a holding register	PIT_B_DISABLE_COUNTER_OVERWRITE, PIT_B_ENABLE_COUNTER_OVERWRITE
PIT_B_DEBUGModeControl_t	DEBUGModeControl	Determines whether the PIT operates in DEBUG mode	ENABLE_PIT_B_IN_DEBUG, DISABLE_PIT_B_IN_DEBUG
PIT_B_DOZEModeControl_t	DOZEModeControl	Determines whether the PIT operates in Doze mode	ENABLE_PIT_B_IN_DOZE, DISABLE_PIT_B_IN_DOZE

PIT_B_PreScalarSelect_t	PreScalarSelect	Determines how the PIT clock is generated from the system clock	PIT_B_SCALAR_00, PIT_B_SCALAR_01, PIT_B_SCALAR_02, PIT_B_SCALAR_03, PIT_B_SCALAR_04, PIT_B_SCALAR_05, PIT_B_SCALAR_06, PIT_B_SCALAR_07, PIT_B_SCALAR_08, PIT_B_SCALAR_09, PIT_B_SCALAR_10, PIT_B_SCALAR_11, PIT_B_SCALAR_12, PIT_B_SCALAR_13, PIT_B_SCALAR_14, PIT_B_SCALAR_15
UINT16	ModulusValue	Data to copy into PMR register.	Any 16-bit value.

11.4.1.4 Return Values

Error Code	Description
PIT_B_ERR_NONE	No error
PIT_B_ERR_INVALID_HANDLE	Device handle is NULL.
PIT_B_ERR_INVALID_COUNTER_RELOAD_CONTROL	Counter reload control is invalid.
PIT_B_ERR_INVALID_COUNTER_OVERWRITE_ENABLE	Counter overwrite enable is invalid.
PIT_B_ERR_INVALID_DEBUG_MODE_CONTROL	DEBUG mode control is invalid.
PIT_B_ERR_INVALID_DOZE_MODE_CONTROL	DOZE mode control is invalid.
PIT_B_ERR_INVALID_PRE_SCALAR_SELECT	Pre-scalar select is invalid.

11.4.2 PIT_B_ControlInterrupt

11.4.2.1 Description

The PIT_B_ControlInterrupt function enables or disables the PIT Interrupt Enable (PIE). Parameter checking is a compile-time option.

This function should only be called after the Programable Interval Timer (PIT) has been initialized through a call to the PIT_B_Init function.

11.4.2.2 Prototype

```
PIT_B_ReturnCode_t PIT_B_ControlInterrupt (
          pPIT_B_t                  PITPtr,
          PIT_B_InterruptEnable_t   PIE,
)
```

11.4.2.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPIT_B_t	PITPtr	PIT base address associated with this driver	Any non-zero core processor data address.
PIT_B_InterruptEnable_t	PIE	Controls pit interrupt enable	PIT_B_INTERRUPT_FLAG_NOT_ALLOWED, PIT_B_INTERRUPT_FLAG_ ALLOWED

11.4.2.4 Return Values

Error Code	Description
PIT_B_ERR_NONE	No error
PIT_B_ERR_INVALID_HANDLE	Device Handle is NULL.
PIT_B_ERR_INVALID_PIT_INTERRUPT_ENABLE	PIT interrupt enable is invalid.

11.4.3 PIT_B_GetStatus

11.4.3.1 Description

Gets the PIT interrupt status by reading the PIT Interrupt Flag (PIF) of the PIT Control/Status Register (PCSR).

This function should only be called after the PIT has been initialized through a call to the PIT_B_Init function.

11.4.3.2 Prototype

```
PIT_B_ReturnCode_t PIT_B_GetStatus (
          pPIT_B_t        PITPtr,
          UINT           *GetStatusPtr
)
```

11.4.3.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPIT_B_t	PITPtr	PIT base address associated with this driver.	Any non-zero core processor data address.
UINT *	GetStatusPtr	Result address for selected PIT_B register data.	Any non-zero core processor data address.

11.4.3.4 Return Values

Error Code	Description
PIT_B_ERR_NONE	No error
PIT_B_ERR_INVALID_HANDLE	PIT base address parameter is zero.
PIT_B_ERR_BAD_RESULT_ADDR	Result pointer is zero.

11.4.4 PIT_B_ControlOperation

11.4.4.1 Description

The PIT_B_ControlOperation function enables or disables the Programmable Interval Timer. Parameter checking is a compile-time option.

This function should only be called after the PIT has been initialized through a call to the PIT_B_Init function.

11.4.4.2 Prototype

```
PIT_B_ReturnCode_t PIT_B_ControlOperation (
          pPIT_B_t              PITPtr,
          PIT_B_PITEnable_t     PITEnable
)
```

11.4.4.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPIT_B_t	PITPtr	PIT base address associated with this driver.	Any non-zero core processor data address.
PIT_B_PITEnable_t	PITEnable	Enables or disables the PIT.	PIT_B_FUNCTION_DISABLE, PIT_B_FUNCTION_ENABLE

11.4.4.4 Return Values

Error Code	Description
PIT_B_ERR_NONE	No error
PIT_B_ERR_INVALID_HANDLE	PIT base address parameter is zero.
PIT_B_ERR_INVALID_PIT_ENABLE	PIT enable is invalid.

11.4.5 PIT_B_SetModulus

11.4.5.1 Description

The PIT_B_SetModulus function shall load a 16-bit data value into the modulus.

This function should only be called after the PIT has been initialized through a call to the PIT_B_Init function.

11.4.5.2 Prototype

```
PIT_B_ReturnCode_t PIT_B_SetModulus (
          pPIT_B_t                          PITPtr,
          UINT16                            Modulus,
          PIT_B_CounterOverwriteEnable_t    CounterOverwriteEnable
)
```

11.4.5.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPIT_B_t	PITPtr	PIT base address associated with this driver.	Any non-zero core processor data address.
UINT16	Modulus	Data to copy into selected PIT_B register.	Any 16-bit value.
PIT_B_CounterOverwriteEnable_t	CounterOverwrite Enable	Determines whether the modulus latch is transparent or a holding register	PIT_B_DISABLE_COUNTER_OVERWRITE, PIT_B_ENABLE_COUNTER_OVERWRITE

11.4.5.4 Return Values

Error Code	Description
PIT_B_ERR_NONE	No error
PIT_B_ERR_INVALID_HANDLE	PIT base address parameter is zero
PIT_B_ERR_INVALID_COUNTER_OVERWRITE_ENABLE	Counter overwrite enable is invalid.

11.4.6 PIT_B_SetRegister

11.4.6.1 Description

The PIT_B_SetRegister function writes a 16-bit data value to a sub-set of Programable Interval Timer registers. Parameter checking is a compile-time option.

This function can be called at any time.

11.4.6.2 Prototype

```
PIT_B_ReturnCode_t PIT_B_SetRegister (
          pPIT_B_t            PITPtr,
          PIT_B_Register_t    RegisterSelect,
          UINT16              RegisterValue
)
```

11.4.6.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPIT_B_t	PITPtr	PIT_B base address associated with this driver.	Any non-zero core processor data address.
PIT_B_REGISTER_t	RegisterSelect	Select Among PIT_B registers.	PIT_B_PCSR, PIT_B_PMR
UINT16	RegisterValue	Data to copy into the user specified register.	Any 16-bit value.

11.4.6.4 Return Values

Error Code	Description
PIT_B_ERR_NONE	No error
PIT_B_ERR_NONE	No error
PIT_B_INVALID_HANDLE	PIT base address parameter is zero
PIT_B_INVALID_REGISTER	PIT register selection switch is invalid

11.4.7 PIT_B_GetRegister

11.4.7.1 Description

The PIT_B_GetRegister function returns the contents of a sub-set of the Programable Interval Timer registers. Parameter checking is a compile-time option

This function can be called at any time.

11.4.7.2 Prototype

```
PIT_B_ReturnCode_t PIT_B_GetRegister (
          pPIT_B_t            PITPtr,
          PIT_B_Register_t    PITRegister,
          UINT16             *GetRegisterPtr
)
```

11.4.7.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPIT_B_t	PITPtr	PIT_B base address associated with this driver.	Any non-zero core processor data address.
PIT_B_Register_t	PITRegister	Select among a sub-set of PIT_B registers.	PIT_B_PCSR, PIT_B_PMR, PIT_B_PCNTR.
UINT *	GetRegisterPtr	Result address for selected PIT_B register data.	Any non-zero core processor data address.

11.4.7.4 Return Values

Error Code	Description
PIT_B_ERR_NONE	No error
PIT_B_INVALID_HANDLE	PIT base address parameter is zero
PIT_B_BAD_RESULT_ADDR	Result Pointer is zero
PIT_B_INVALID_REGISTER	PIT Register Selection switch is invalid

11.4.8 PIT_B_Reset

11.4.8.1 Description

The PIT_B_Reset function resets the Programmable Interval Timer (PIT). Parameter checking is a compile-time option.

This function should be called once during the system software initialization.

11.4.8.2 Prototype

```
PIT_B_ReturnCode_t  PIT_B_Reset (
          pPIT_B_t    PITPtr
)
```

11.4.8.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPIT_B_t	PITPtr	PIT base address associated with this driver	Any non-zero core processor data address.

11.4.8.4 Return Values

Error Code	Description
PIT_B_ERR_NONE	No error
PIT_B_ERR_INVALID_HANDLE	Device handle is NULL.
PIT_B_ERR_INVALID_COUNTER_OVERWRITE_ENABLE	Counter overwrite enable is invalid.

11.5 PIT_B Example Application

The following describes the PIT_B example application.

11.5.1 Description for Example Application

The example application initializes the programmable timer, demonstrates how to start and stop the counter, and sets a new timer modulus.

11.5.1.1 Example Application Code

```
/*********************************************************************

File:  pitb_app.c

Purpose:The following example code illustrates the use of the M-CORE
        Peripherals Library PIT_B Level 1 driver.

        The program initializes the pit and then disables the PIT.
        It sets the modulus to 0xA5, loads it and enables the PIT. The
        PIT should be counting down to 0x0, and then rolls over to
        0xA5. After waiting for a period of time, the PIT is disabled.
        GetRegister is called to get the value of the counter.

        If an error occurs, the program will exit with the line number
        of where the failure occurred.

Time to run:    About 5 seconds.

General Setup:
        General hardware and software setup to run this application can
        be found in the 'readme_app.txt' file, located in the
        '<library>/app' directory.

Application Specific Setup:
        Developement Board: MMC2107

(C) Copyright Metrowerks Inc, 2000.  All rights reserved.
---------------------------------------------------------------------*/

#include "pit_b.h"
#include "plibdefs_mmc2107.h"

/* macro to handle error checking and response                       */
#define handle_error(err) if (err != PIT_B_ERR_NONE) return(__LINE__);

int main(void)
{
  pPIT_B_t pitptr = ((pPIT_B_t)__MMC2107_PIT1);
  PIT_B_ReturnCode_t returnValue;
  UINT16 ModulusValue = 0xA5;
  UINT16 j = 0;
```

```
UINT16 value;

/* Initialize the PIT.  If an error occurs, exit the program        */
returnValue = PIT_B_Init(pitptr,
                     /* PIT base address associated with this driver*/
                         PIT_B_COUNTER_RELOAD_FROM_MODULUS,
                     /* PIT is reloaded from modulus value          */
                         PIT_B_DISABLE_COUNTER_OVERWRITE,
                     /* Modulus latch is transparent                */
                         PIT_B_ENABLE_IN_DEBUG,
                     /* PIT operates in DEBUG                       */
                         PIT_B_ENABLE_IN_DOZE,
                     /* PIT operates in DOZE                        */
                         PIT_B_SCALAR_00,
                     /* PIT clock is the same as the system clock   */
                         ModulusValue
                     /* Data to copy into PIT_PMR register          */
                        );

handle_error(returnValue);

/* Disable the PIT.  If an error occurs, exit the program           */
returnValue = PIT_B_ControlOperation(pitptr, /* PIT base address    */
                                     PIT_B_FUNCTION_DISABLE
                                             /* Disable the PIT     */
                                     );

handle_error(returnValue);

/* Set the PIT modulus.  If the error occurs, exit the program      */
returnValue = PIT_B_SetModulus(pitptr, /* PIT base address          */
                               ModulusValue,/* PIT modulus value    */
                               PIT_B_DISABLE_COUNTER_OVERWRITE
                               );

handle_error(returnValue);

/* Enable the PIT.  If an error occurs, exit the program            */
returnValue = PIT_B_ControlOperation(pitptr, /* PIT base address    */
                                     PIT_B_FUNCTION_ENABLE
                                     /* Enable the PIT              */
                                     );

handle_error(returnValue);

/* Wait for a period of time to allow the PIT to count */
while(j < 20000)
{
  j++;
}

/* Disable the PIT.  If an error occurs, exit the program           */
returnValue = PIT_B_ControlOperation(pitptr, /* PIT base address    */
                                     PIT_B_FUNCTION_DISABLE
```

PIT_B Level 1

```
                                    /* Disable the PIT              */
                                     );

  handle_error(returnValue);

  /* Get the register contents of PCNTR.  If an error occures,      */
  /* exit the program                                               */
  returnValue = PIT_B_GetRegister(pitptr,      /* PIT base address    */
                                  PIT_B_PCNTR, /* Select the PCNTR to */
                                               /* read                */

                                  &value     /* The contents of PCNTR */
                                  );

  handle_error(returnValue);


  return(0);

} /* end of main */
```

Section 12 PLL_B Level 1

This section describes the PLL_B Level 1 M•CORE™ Device Driver.

12.1 Overview

The Phase Lock Loop (PLL) and its accompanying driver (PLL_B) are used on the MMC2107 development system. The level 1 service is designed to allow the programmer to control the PLL on a near-register level, but without the concern of exact positions of registers and fields, or the detailed sequences of register operations. Refer to the MMC2107 Advance Information (Data Book) (MMC2107/D) for additional information.

12.2 PLL_B Service Functions

The PLL_B device driver software provides these functions:

- Initializes the Phase Lock Loop module.
- Returns the PLL status.
- Resets the synthesizer control register (SYNCR) to the factory default values.
- Gets the PLL Runtime status.
- Controls the frequency at which the clock out frequency will be set.
- Controls the actions that will be taken in the event of Clock failure.
- Controls the actions that take place when the system goes into the Stop mode.
- Enables or disables the Clockout signal.
- Gets unformatted data from specified PLL_B registers.
- Writes 16-bit values to the PLL_B registers.

12.3 PLL_B Data Type Definitions

The following paragraphs describe the data definitions recognized by the PLL_B device driver. These data definitions are grouped into six categories: standard data types, board specific base address definitions, derived data types, enumerated data types, register macros, and return codes.

12.3.1 Standard Data Type Definitions

- UINT8 — 8-bit unsigned data value
- INT8 — 8-bit signed data value

- UINT16 — 16-bit unsigned data value
- INT16 — 16-bit signed data value
- UINT32 — 32-bit unsigned data value
- INT32 — 32-bit signed data value

12.3.2 Board Specific Base Address Definitions

```
#define __MMC2107_PLL   (0x00C3_0000)
```

12.3.3 Derived Data Type Definitions

12.3.3.1 PLL_B_t — PLL_B Register Definition

```
typedef struct {
           UINT16    SYNCR;    /* PLL Control registers */
   Volatile UINT16   SYNSR;    /* PLL Status Register   */
} PLL_B_t, *pPLL_B_t;
```

12.3.3.2 PLL_B_Status_t — PLL_B_GetResetStatus Stucture Definition

```
typedef struct {
      PLL_B_ClockMode_t       ClockMode;      /* PLL Clock operation mode        */
      PLL_B_PLLMode_t         PLLMode;        /* PLL mode select                 */
      PLL_B_ClockSource_t     ClockSource;    /* External Clock reference source */
      PLL_B_StickyLockBit_t   StickyLockBit;  /* Loss of Lock Sticky Bit         */
      PLL_B_LockStatus_t      LockStatusBit;  /* PLL Lock status bit             */
      PLL_B_StickyCLockBit_t  StickyClockBit; /* Loss of Clock Sticky Bit        */
}  PLL_B_Status_t, *pPLL_B_Status_t;
```

12.3.3.3 PLL_B_StatusAlert_t — PLL_B_StatusAlert Structure Definition

```
typedef struct {
      PLL_B_Boolean_t    InvalidPLLMode;   /* Wrong mode to change frequency */
      PLL_B_Boolean_t    PhaseLockFailure; /* Failure to obtain phase lock   */
} PLL_B_StatusAlert_t, *pPLL_B_StatusAlert_t;
```

12.3.4 Enumerated Data Type Definitions

12.3.4.1 PLL_B_Register_t – PLL_B Register Selection

```
typedef enum {
    PLL_B_SYNCR,      /* PLL Control registers */
    PLL_B_SYNSR       /* PLL Status Register   */
} PLL_B_Register_t;
```

12.3.4.2 PLL_B_StatusAlert_t – PLL_B PLL Mode Status warning

```
typedef enum {
    PLL_B_VALID_MODE,
    PLL_B_INVALID_MODE
} PLL_B_Mode_t;
```

12.3.4.3 PLL_B_Status enum definitions

12.3.4.3.1 PLL_B_ClockMode_t — Clock Operations Mode

```
typedef enum {
      PLL_B_EXTERNAL,
      PLL_B_PLL
} PLL_B_ClockMode_t;
```

12.3.4.3.2 PLL_B_PLLMode_t — PLL Mode

```
typedef enum {
      PLL_B_1_TO_1_PLL_MODE,
      PLL_B_NORMAL_PLL_MODE
} PLL_B_PLLMode_t;
```

12.3.4.3.3 PLL_B_ClockSource_t — PLL Clock Reference Source

```
typedef enum {
     PLL_B_CLOCK_SOURCE,
     PLL_B_CRYSTAL_SOURCE
} PLL_B_ClockSource_t;
```

12.3.4.3.4 PLL_B_StickyLockBit_t — Loss of Lock Sticky bit

```
typedef enum {
     PLL_B_HAS_LOST_LOCK,
     PLL_B_HAS_NOT_LOST_LOCK
} PLL_B_StickyLockBit_t;
```

12.3.4.3.5 PLL_B_LockStatus_t — Lock Status bit

```
typedef enum {
     PLL_B_UNLOCKED,
     PLL_B_LOCKED
} PLL_B_LockStatus_t;
```

12.3.4.3.6 PLL_B_StickyClockBit_t — Loss of Clock Sticky bit

```
typedef enum {
     PLL_B_HAS_NOT_LOST_CLOCK,
     PLL_B_HAS_LOST_CLOCK
} PLL_B_StickyClockBit_t;
```

12.3.4.4 PLL_B_ReturnCode_t – PLL_D Error Codes

```
typedef enum {
 PLL_B_ERR_NONE,                      /* no error                                  */
 PLL_B_ERR_INVALID_HANDLE,            /* PLL base address parameter is zero        */
 PLL_B_ERR_BAD_RESULT_ADDR,           /* Result pointer is zero                    */
 PLL_B_ERR_INVALID_REGISTER,          /* Register selection if invalid             */
 PLL_B_ERR_INVALID_LOSS_LOCK_RESET,   /* Reset on loss of lock is invalid          */
 PLL_B_ERR_INVALID_FREQUENCY_MULTIPLIER, /* Frequency multiplier is invalid        */
 PLL_B_ERR_INVALID_LOSS_CLOCK_RESET,  /* Reset on loss of clock is invalid         */
 PLL_B_ERR_INVALID_FREQUENCY_DIVIDER, /* Frequency divider is invalid              */
 PLL_B_ERR_INVALID_MONITOR_CLOCK,     /* Enable monitor loss of clock is invalid */
 PLL_B_ERR_INVALID_CLOCK_OUT_SIGNAL,  /*Enable/disable clockout signal is invalid*/
 PLL_B_ERR_INVALID_FAST_WAKEUP,       /* Fast wakeup is invalid                    */
 PLL_B_ERR_INVALID_STOP_MODE          /* Stop mode selection is invalid            */
 PLL_B_ERR_INVALID_STATUS_ALERT_ADDR  /* Bad Status alert pointer address          */
} PLL_B_ReturnCode_t;
```

12.3.4.5 PLL_B Control Register enum Definitions

12.3.4.5.1 PLL_B_ResetOnLockFailureControl — Control Selection

```
typedef enum  {
  PLL_B_RESET_ON_LOCK_FAILURE_DISABLE, /* disable reset on loss of lock selection*/
  PLL_B_RESET_ON_LOCK_FAILURE_ENABLE   /* enable reset on loss of lock selection */
} PLL_B_ResetOnLockFailureControl_t;
```

12.3.4.5.2 PLL_B_VCO_MultiplyFactorDivider selection

```
typedef enum {
   PLL_B_VCO_X_2,      /* Set VCO clockout to 2 times the reference frequency */
   PLL_B_VCO_X_3,      /* 3 times  */
   PLL_B_VCO_X_4,      /* 4 times  */
   PLL_B_VCO_X_5,      /* 5 times  */
   PLL_B_VCO_X_6,      /* 6 times  */
   PLL_B_VCO_X_7,      /* 7 times  */
   PLL_B_VCO_X_8,      /* 8 times  */
   PLL_B_VCO_X_9,      /* 9 times  */
   PLL_B_VCO_X_NO_CHANGE
} PLL_B_VCO_MultiplyFactorDivider_t;
```

12.3.4.5.3 PLL_B_ResetOnClockFailureControl_t — Control Selection

```
typedef enum  {
 PLL_B_RESET_ON_CLOCK_FAILURE_DISABLE, /*disable reset on loss of clock selection*/
 PLL_B_RESET_ON_CLOCK_FAILURE_ENABLE   /*enable reset on loss of clock selection */
} PLL_B_ResetOnClockFailureControl_t;
```

12.3.4.5.4 PLL_B_VCO_ReducedFrequencyDivider Selection

```
typedef enum  {
PLL_B_VCO_DIV_1, /* Establish the divisor of 1 to be applied to the PLL frequency*/
PLL_B_VCO_DIV_2,               /* divisor of 2      */
PLL_B_VCO_DIV_4,               /* divisor of 4      */
PLL_B_VCO_DIV_8,               /* divisor of 8      */
PLL_B_VCO_DIV_16,              /* divisor of 16     */
PLL_B_VCO_DIV_32,              /* divisor of 32     */
PLL_B_VCO_DIV_64,              /* divisor of 64     */
PLL_B_VCO_DIV_128,             /* divisor of 128    */
PLL_B_VCO_DIV_NO_CHANGE
} PLL_B_VCO_ReducedFrequencyDivider_t;
```

12.3.4.5.5 PLL_B_MonitorClockFailure_Control_t — Control Selection

```
typedef enum  {
 PLL_B_MONITOR_CLOCK_FAILURE_DISABLE, /* disable reset on loss of clock selection*/
 PLL_B_MONITOR_CLOCK_FAILURE_ENABLE   /* enable reset on loss of clock selection */
} PLL_B_MonitorClockFailure_t;
```

12.3.4.5.6 PLL_B_DisableClockOut_t — Control Selection

```
typedef enum  {
    PLL_B_CLOCK_OUT_ENABLE,  /* enable output of VCO Clockout signal selection  */
    PLL_B_CLOCK_OUT_DISABLE  /* disable output of VCO Clockout signal selection */
} PLL_B_DisableClockOut_t;
```

12.3.4.5.7 PLL_B_FastWakeupControl_t — Control Selection

```
typedef enum {
   PLL_B_FAST_WAKEUP_DISABLE, /* disable fast wakeup from stop mode selection */
   PLL_B_FAST_WAKEUP_ENABLE   /* enable fast wakeup from stop mode selection  */
} PLL_B_FastWakeupControl_t;
```

12.3.4.5.8 PLL_B_StopModeControl_t — Control Selection

```
typedef enum  {
    PLL_B_ENABLE_PLL_OSC_CLKOUT,  /* Enable all options during stop mode     */
    PLL_B_ENABLE_OSC,             /* Enable only the OSC during stop         */
    PLL_B_ENABLE_PLL_OSC,         /* Enable only the PLL and OSC during stop */
    PLL_B_DISABLE_ALL
} PLL_B_StopModeControl_t;
```

12.3.5 Register Macros

12.3.5.1 Synthesizer Control Register

```
#define SYNCR_STPMD_MAX   0x3
#define SYNCR_STPMD_BITNO 2
#define SYNCR_STPMD_MASK (SYNCR_STPMD_MAX << SYNCR_STPMD_BITNO) /* 0000_0000_0000_1100*/

#define SYNCR_FWKUP_BITNO 5
#define SYNCR_FWKUP_MASK (1 << SYNCR_FWKUP_BITNO)  /* 0000_0000_0010_0000  */
#define SYNCR_DISCLK_BITNO 6
#define SYNCR_DISCLK_MASK (1 << SYNCR_DISCLK_BITNO) /* 0000_0000_0100_0000 */

#define SYNCR_LOCEN_BITNO  7
#define SYNCR_LOCEN_MASK (1 << SYNCR_LOCEN_BITNO)   /* 0000_0000_1000_0000 */

#define SYNCR_RFD_MAX    0x7          /* 0000_0111_0000_0000 */
#define SYNCR_RFD_RESET  0x1
#define SYNCR_RFD_BITNO  8
#define SYNCR_RFD_MASK  (SYNCR_RFD_MAX << SYNCR_RFD_BITNO)/* 0000_0001_0000_0000 */

#define SYNCR_LOCRE_BITNO  11
#define SYNCR_LOCRE_MASK (1 << SYNCR_LOCRE_BITNO)   /* 0000_1000_0000_0000 */

#define SYNCR_MFD_MAX    0x7          /* 0111_0000_0000_0000 */
#define SYNCR_MFD_RESET  0x2
#define SYNCR_MFD_BITNO  12
#define SYNCR_MFD_MASK  (SYNCR_MFD_MAX << SYNCR_MFD_BITNO)  /* 0010_0000_0000_0000 */

#define SYNCR_LOLRE_BITNO 15
#define SYNCR_LOLRE_MASK (1 << SYNCR_LOLRE_BITNO)    /* 1000_0000_0000_0000 */

#define SYNCR_RFD_RESET_MASK  (SYNCR_RFD_RESET << SYNCR_RFD_BITNO)
#define SYNCR_MFD_RESET_MASK  (SYNCR_MFD_RESET << SYNCR_MFD_BITNO)
#define SYNCR_RESET_MASK  (SYNCR_RFD_RESET_MASK + SYNCR_MFD_RESET_MASK)
```

12.3.5.2 Status Register

```
#define SYNSR_LOCS_BITNO   10
#define SYNSR_LOCS_MASK     (1 << SYNSR_LOCS_BITNO)   /* 0000_0100_0000_0000 */
#define SYNSR_LOCK_BITNO   11
#define SYNSR_LOCK_MASK     (1 << SYNSR_LOCK_BITNO)   /* 0000_1000_0000_0000 */
#define SYNSR_LOCKS_BITNO  12
#define SYNSR_LOCKS _MASK   (1 << SYNSR_LOCKS_BITNO)  /* 0001_0000_0000_0000 */
#define SYNSR_PLLREF_BITNO 13
#define SYNSR_PLLREF_MASK   (1 << SYNSR_PLLREF_BITNO) /* 0010_0000_0000_0000 */
#define SYNSR_PLLSEL_BITNO 14
#define SYNSR_PLLSEL_MASK   (1 << SYNSR_PLLSEL_BITNO) /* 0100_0000_0000_0000 */
#define SYNSR_MODE_BITNO   15
#define SYNSR_MODE_MASK     (1 << SYNSR_MODE_BITNO)   /* 1000_0000_0000_0000 */
```

12.3.6 Return Codes

Return codes from the enumerated type “PLL_B_ReturnCode”.

Return Code	Functions Returning	Description
PLL_B_ERR_NONE	All	No error.
PLL_B_ERR_INVALID_HANDLE	All	PLL base address parameter is zero.
PLL_B_ERR_BAD_RESULT_ADDR	PLL_B_GetResetStatus, PLL_B_GetRuntimeStatus, PLL_B_GetRegister	Result Pointer is zero.
PLL_B_ERR_INVALID_REGISTER	PLL_B_GetRegisrer, PLL_B_SetRegister	PLL Register selection switch is invalid
PLL_B_ERR_INVALID_LOSS_LOCK_RESET	PLL_B_Init, PLL_B_ControlClockFailureResponse	Reset on loss of lock is invalid
PLL_B_ERR_INVALID_LOSS_CLOCK_RESET	PLL_B_Init, PLL_B_ControlClockFailureResponse	Reset on loss of clock is invalid
PLL_B_ERR_INVALID_FREQUENCY_MULTIPLIER	PLL_B_Init, PLL_B_ControlVCOFrequency	Frequency multiplier is invalid
PLL_B_ERR_INVALID_FREQUENCY_DIVIDER	PLL_B_Init, PLL_B_ControlVCOFrequency	Frequency divider is invalid
PLL_B_ERR_INVALID_MONITOR_CLOCK	PLL_B_Init, PLL_B_ControlClockFailureResponse	Enable monitor loss of clock is invalid
PLL_B_ERR_INVALID_CLOCK_OUT_SIGNAL	PLL_B_Init, PLL_B_ControlVCOOutput	Enable/disable clockout signal is invalid
PLL_B_ERR_INVALID_FAST_WAKEUP	PLL_B_Init, PLL_B_ControlStopMode	Fast wakeup is invalid
PLL_B_ERR_INVALID_STOP_MODE	PLL_B_Init, PLL_B_ControlStopMode	Stop mode selection is invalid
PLL_B_ERR_INVALID_STATUS_ALERT_ADDR	PLL_B_Init, PLL_B_ControlVCOFrequency	Null pointer for Status Alert Warnings

12.4 PLL_B API Definitions

12.4.1 PLL_B_Init

12.4.1.1 Description

The PLL_B_Init function initializes the Phase Lock Loop (PLL) module. Parameter checking is a compile-time option.

This function should be called during the system software initialization or for re-initialization of the module.

12.4.1.2 Prototype

```
PLL_B_ReturnCode_t PLL_B_Init (
          pPLL_B_t                              PLLPtr,
          PLL_B_ResetOnLockFailureControl_t     ResetOnLockFailureValue,
          PLL_B_VCOMultiplyFactorDivider_t      VCOMultiplicationFactorDivider,
          PLL_B_ResetOnClockFailureControl_t    ResetOnClockFailureValue,
          PLL_B_VCOReducedFrequencyDivider_t    VCOReducedFrequencyDivider,
          PLL_B_MonitorClockFailure_t           MonitorClockFailureValue,
          PLL_B_DisableClockOut_t               DisableClockoutValue,
          PLL_B_FastWakeupControl_t             FastWakeupValue,
          PLL_B_StopModeControl_t               StopModeValue,
          pPLL_B_StatusAlert_t                  StatusAlertPtr,
          pPLL_B_TimeoutFunction_t              TimeoutFunctionPtr
)
```

12.4.1.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPLL_B_t	PLLPtr	PLL_B base address associated with this driver.	Any non-zero core processor data address.
PLL_B_ResetOnLock FailureControl_t	ResetOnLockFailureValue	Enable or Disable the Reset on loss of lock function	PLL_B_RESET_ON_LOCK_ FAILURE_DISABLE, PLL_B_RESET_ON_LOCK_ FAILURE_ENABLE
PLL_B_VCOMultiplyF actorDivider_t	VCOMultiplicationFactorDivider	Frequency out multiplier value	PLL_B_VCO_X_2, PLL_B_VCO_X_3, PLL_B_VCO_X_4, PLL_B_VCO_X_5, PLL_B_VCO_X_6, PLL_B_VCO_X_7, PLL_B_VCO_X_8, PLL_B_VCO_X_9, PLL_B_VCO_X_NO_CHANGE
PLL_B_ResetOnClock FailureControl_t	ResetOnClockFailureValue	Enable or Disable the Reset on loss of clock function	PLL_B_RESET_ON_CLOCK_ FAILURE_DISABLE, PLL_B_RESET_ON_CLOCK_ FAILURE_ENABLE
PLL_B_VCOReduced FrequencyDivider_t	VCOReducedFrequencyDivider	Frequency out divider value	PLL_B_VCO_DIV_1, PLL_B_VCO_DIV_2, PLL_B_VCO_DIV_4, PLL_B_VCO_DIV_8, PLL_B_VCO_DIV_16, PLL_B_VCO_DIV_32, PLL_B_VCO_DIV_64, PLL_B_VCO_DIV_128, PLL_B_VCO_DIV_NO_CHANGE
PLL_B_MonitorClockF ailure_t	MonitorClockFailureValue	Enable or Disable the loss of lock monitor function	PLL_B_MONITOR_CLOCK_ FAILURE_DISABLE, PLL_B_MONITOR_CLOCK_ FAILURE_ENABLE
PLL_B_DisableClock Out_t	DisableClockoutValue	Enable or Disable the CLKOUT signal	PLL_B_CLOCK_OUT_ENABLE, PLL_B_CLOCK_OUT_DISABLE
PLL_B_FastWakeupC ontrol_t	FastWakeupValue	Enable or Disable the Fast Wakeup function	PLL_B_FAST_WAKEUP_DISABLE, PLL_B_FAST_WAKEUP_ENABLE
PLL_B_StopModeCon trol_t	StopModeValue	Enable or Disable the PLL, OSC or CLKOUT signal during Stop the mode	PLL_B_ENABLE_PLL_OSC_ CLKOUT, PLL_B_ENABLE_PLL_OSC, PLL_B_ENABLE_OSC
pPLL_B_StatusAlert_t	StatusAlertPtr	Structure for Status Alert fields	Any non-zero core processor address
pPLL_B_TimeoutFunc tion_t	TimeoutFunctionPtr	Pointer to Timeout Function for lock bit polling	Any non-zero core processor address

12.4.1.4 Return Values

Error Code	Description
PLL_B_ERR_NONE	No error
PLL_B_ERR_NONE	No error.
pPLL_B_t	PLLPtr
PLL_B_ResetOnLockFailureControl_t	ResetOnLockFailureValue
PLL_B_VCOMultiplyFactorDivider_t	VCOMultiplicationFactorDivider
PLL_B_ResetOnClockFailureControl_t	ResetOnClockFailureValue
PLL_B_VCOReducedFrequencyDivider_t	VCOReducedFrequencyDivider
PLL_B_MonitorClockFailure_t	MonitorClockFailureValue
PLL_B_DisableClockOut_t	DisableClockoutValue
PLL_B_FastWakeupControl_t	FastWakeupValue
PLL_B_StopModeControl_t	StopModeValue
pPLL_B_StatusAlert_t	StatusAlertPtr

12.4.2 PLL_B_GetResetStatus

12.4.2.1 Description

The PLL_B_GetResetStatus function returns the PLL status. Parameter checking is a compile-time option.

This function is designed to be called directly after a call to the PLL_B_Reset function.

12.4.2.2 Prototype

```
PLL_B_ReturnCode_t  PLL_B_GetResetStatus  (
          pPLL_B_t           PLLPtr,
          pPLL_B_Status_t    StatusPtr
)
```

12.4.2.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPLL_B_t	PLLPtr	PLL base address associated with this driver.	Any non-zero core processor data address.
pPLL_B_Status_t	StatusPtr	Pointer to result data address	Any non-zero memory address

12.4.2.4 Return Values

Error Code	Description
PLL_B_ERR_NONE	No error
PLL_B_ERR_INVALID_HANDLE	PLL base address parameter is zero
PLL_B_ERR_BAD_RESULT_ADDR	User supplied result address is zero

12.4.3 PLL_B_Reset

12.4.3.1 Description

The PLL_B_Reset function resets the synthesizer control register (SYNCR) to the factory default values. Parameter checking is a compile-time option.

This function can be called at any time.

12.4.3.2 Prototype

```
PLL_B_ReturnCode_t  PLL_B_Reset (
          pPLL_B_t    PLLPtr
)
```

12.4.3.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPLL_B_t	PLLPtr	PLL base address associated with this driver.	Any non-zero core processor data address.

12.4.3.4 Return Values

Error Code	Description
PLL_B_ERR_NONE	No error
PLL_B_ERR_INVALID_HANDLE	PLL base address parameter is zero

12.4.4 PLL_B_GetRuntimeStatus

12.4.4.1 Description

The PLL_B_GetRuntimeStatus function returns the PLL Runtime status. Parameter checking is a compile-time option.

This function can be call at anytime.

12.4.4.2 Prototype

```
PLL_B_ReturnCode_t  PLL_B_GetRuntimeStatus  (
          pPLL_B_t           PLLPtr,
          pPLL_B_Status_t    StatusPtr
)
```

12.4.4.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPLL_B_t	PLLPtr	PLL base address associated with this driver.	Any non-zero core processor data address.
pPLL_B_Status_t	StatusPtr	Pointer to result data address	Any non-zero memory address

12.4.4.4 Return Values

Error Code	Description
PLL_B_ERR_NONE	No error
PLL_B_ERR_INVALID_HANDLE	PLL base address parameter is zero
PLL_B_ERR_BAD_RESULT_ADDR	User supplied result address is zero

12.4.5 PLL_B_ControlVCOFrequency

12.4.5.1 Description

The PLL_B_ControlVCOFrequency function controls the frequency at which the clock out frequency is be set. Parameter checking is a compile time option.

The PLL_B_ControlVCOFrequency function should only be called when the PLL is in the NORMAL PLL mode.

12.4.5.2 Prototype

```
PLL_B_ReturnCode_t  PLL_B_ControVCOFrequency (
          pPLL_B_t   PLLPtr,
          PLL_B_VCOMultiplyFactorDivider_t      VCOMultiplicationFactorDivider,
          PLL_B_VCOReducedFrequencyDivider_t    VCOReducedFrequencyDivider,
          pPLL_B_TimeoutFunction_t              TimeoutFunctionPtr,
          pPLL_B_StatusAlert_t                  StatusAlertPtr
)
```

12.4.5.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPLL_B_t	PLLPtr	PLL base address associated with this driver.	Any non-zero core processor data address.
PLL_B_VCOMultiplyFactorDivider_t	VCOMultiplicationFactorDivider	Select among the MFD values.	PLL_B_VCO_X_2, PLL_B_VCO_X_3, PLL_B_VCO_X_4, PLL_B_VCO_X_5, PLL_B_VCO_X_6, PLL_B_VCO_X_7, PLL_B_VCO_X_8, PLL_B_VCO_X_9, PLL_B_VCO_X_NO_CHANGE
PLL_B_VCOReducedFrequencyDivider_t	VCOReducedFrequencyDivider	Select among the RFD values.	PLL_B_VCO_DIV_1, PLL_B_VCO_DIV_2, PLL_B_VCO_DIV_4, PLL_B_VCO_DIV_8, PLL_B_VCO_DIV_16, PLL_B_VCO_DIV_32, PLL_B_VCO_DIV_64, PLL_B_VCO_DIV_128, PLL_B_VCO_DIV_NO_CHANGE
pPLL_B_StatusAlert_t	StatusAlert	Structure for Status Alert fields	Any non-zero core processor address
pPLL_B_TimeoutFunction_t	TimeoutFunctionPtr	Pointer to Timeout Function for lock bit polling	Any non-zero core processor address

12.4.5.4 Return Values

Error Code	Description
PLL_B_ERR_NONE	No error
PLL_B_ERR_INVALID_HANDLE	PLL base address parameter is zero
PLL_B_ERR_INVALID_FREQUENCY_MULTIPLIER	Not a valid control selection
PLL_B_ERR_INVALID_FREQUENCY_DIVIDER	Not a valid control selection
PLL_B_ERR_INVALID_STATUS_ALERT_ADDR	Invalid pointer to Status Alert Structure

12.4.6 PLL_B_ControlClockFailureResponse

12.4.6.1 Description

The PLL_B_ControlClockFailureResponse function controls the actions that will be taken in the event of a Clock failure. Parameter checking is a compile-time option.

This function should only be called after the synthesizer control register (SYNCR) has been reset through a previous call to the PLL_B_Reset function.

PLL_B Level 1

12.4.6.2 Prototype

```
PLL_B_ReturnCode_t  PLL_B_ControlClockFailureResponse (
          pPLL_B_t                              PLLPtr,
          PLL_B_ResetOnLockFailureControl_t     ResetOnLockFailureValue,
          PLL_B_ResetOnClockFailureControl_t    ResetOnClockFailureValue,
          PLL_B_MonitorClockFailure_t           MonitorClockFailureValue
)
```

12.4.6.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPLL_B_t	PLLPtr	PLL_B base address associated with this driver.	Any non-zero core processor data address.
PLL_B_ResetOnLockFailureControl_t	ResetOnLockFailureValue	Enable or Disable the Reset on loss of lock function	PLL_B_RESET_ON_LOCK_FAILURE_DISABLE, PLL_B_RESET_ON_LOCK_FAILURE_ENABLE
PLL_B_ResetOnClockFailureControl_t	ResetOnClockFailureValue	Enable or Disable the Reset on loss of clock function	PLL_B_RESET_ON_CLOCK_FAILURE_DISABLE, PLL_B_RESET_ON_CLOCK_FAILURE_ENABLE
PLL_B_MonitorClockFailure_t	MonitorClockFailureValue	Enable or Disable the loss of lock monitor function	PLL_B_MONITOR_CLOCK_FAILURE_DISABLE, PLL_B_MONITOR_CLOCK_FAILURE_ENABLE

12.4.6.4 Return Values

Error Code	Description
PLL_B_ERR_NONE	No error
PLL_B_ERR_INVALID_HANDLE	PLL base address parameter is zero.
PLL_B_ERR_INVALID_LOSS_LOCK_RESET	Reset on loss of lock selection is invalid
PLL_B_ERR_INVALID_LOSS_CLOCK_RESET	Reset on loss of lock selection is invalid
PLL_B_ERR_INVALID_MONITOR_CLOCK	Monitor loss of clock selection is invalid

12.4.7 PLL_B_ControlStopMode

12.4.7.1 Description

The PLL_B_ControlStopMode function controls the actions that take place when the system goes into the Stop mode. Parameter checking is a compile-time option.

This function should only be called after the synthesiser control register (SYNCR) has been reset through a previous call to the PLL_B_Reset function.

12.4.7.2 Prototype

```
PLL_B_ReturnCode_t  PLL_B_ControlStopMode (
          pPLL_B_t                      PLLPtr,
          PLL_B_FastWakeupControl_t     FastWakeupValue,
          PLL_B_StopModeControl_t       StopModeValue
)
```

12.4.7.3 Arguments

Data Type	Formal Name	Description	Valid Values
PLL_B_t	PLLPtr	PLL_B base address associated with this driver.	Any non-zero core processor data address.
PLL_B_FastWakeupControl_t	FastWakeupValue	Enable or Disable the Fast Wakeup function	PLL_B_FAST_WAKEUP_DISABLE, PLL_B_FAST_WAKEUP_ENABLE
PLL_B_StopModeControl_t	StopModeValue	Enable or Disable the PLL, OSC or CLKOUT signal during Stop the mode	PLL_B_ENABLE_PLL_OSC_CLKOUT, PLL_B_ENABLE_PLL_OSC, PLL_B_ENABLE_OSC

12.4.7.4 Return Values

Error Code	Description
PLL_B_ERR_NONE	No error
PLL_B_ERR_INVALID_HANDLE	PLL base address parameter is zero.
PLL_B_ERR_INVALID_FAST_WAKEUP	Fast wake up out of stop mode is invalid
PLL_B_ERR_INVALID_STOP_MODE	Invalid stop mode selection

12.4.8 PLL_B_ControlVCOOutput

12.4.8.1 Description

The PLL_B_ControlVCOOutput function enables or disables the Clockout signal. Parameter checking is a compile-time option.

This function should only be called after the synthesizer control register (SYNCR) has been reset through a previous call to the PLL_B_Reset function.

12.4.8.2 Prototype

```
PLL_B_ReturnCode_t  PLL_B_ControlVCOOutput (
          pPLL_B_t                  PLLPtr,
          PLL_B_DisableClockout_t   DisableClockoutValue
)
```

12.4.8.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPLL_B_t	PLLPtr	PLL_B base address associated with this driver.	Any non-zero core processor data address.
PLL_B_DisableClockout_t	DisableClockoutValue	Enable or Disable the CLKOUT signal.	PLL_B_CLOCK_OUT_ENABLE, PLL_B_CLOCK_OUT_DISABLE

12.4.8.4 Return Values

Error Code	Description
PLL_B_ERR_NONE	No error
PLL_B_ERR_INVALID_HANDLE	PLL base address parameter is zero.
PLL_B_ERR_INVALID_CLOCK_OUT_SIGNAL	Invalid DisableClockoutValue value

12.4.9 PLL_B_GetRegister

12.4.9.1 Description

The PLL_B_GetRegister function reads unformatted data from specified PLL_B registers. Parameter checking is a compile-time option.

This function can be called at any time.

12.4.9.2 Prototype

```
PLL_B_ReturnCode_t PLL_B_GetRegister (
          pPLL_B_t             PLLPtr,
          PLL_B_Register_t     Register,
          UINT16              *GetRegisterPtr
)
```

12.4.9.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPLL_B_t	PLLPtr	PLL_B base address associated with this driver.	Any non-zero core processor data address.
PLL_B_Register_t	Register	Select among PLL_B registers.	PLL_B_SYNCR, PLL_B_SYNSR
UINT16 *	GetRegisterPtr	Result address for selected PLL_B register.	Any non-zero core processor data address.

12.4.9.4 Return Values

Error Code	Description
PLL_B_ERR_NONE	No error
PLL_B_ERR_INVALID_HANDLE	PLL base address parameter is zero
PLL_B_ERR_INVALID_REGISTER	PLL Register Selection switch is invalid
PLL_B_ERR_BAD_RESULT_ADDR	Result pointer is zero

12.4.10 PLL_B_SetRegister

12.4.10.1 Description

The PLL_B_SetRegister function writes a 16-bit value to a PLL_B register. Parameter checking is a compile-time option.

This function can be called at any time.

12.4.10.2 Prototype

```
PLL_B_ReturnCode_t  PLL_B_SetRegister (
          pPLL_B_t    PLLPtr,
          UINT16      SetRegisterValue
)
```

12.4.10.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPLL_B_t	PLLPtr	PLL_B base address associated with this driver.	Any non-zero core processor data address.
UINT16	SetRegisterValue	Data to copy into selected register.	Any 16 bit data value

12.4.10.4 Return Values

Error Code	Description
PLL_B_ERR_NONE	No error
PLL_B_ERR_INVALID_HANDLE	PLL_B base parameter is zero.

12.5 PLL_B Example Application

The following describes the PLL_B example application.

12.5.1 Description for Example Application

The example application illustrates the use of the M-CORE Peripherals Library PLL_B Level 1 driver. The program first obtains the reset status. The program then initializes using the frequency of 16mhz, and disables the VCO Clockout signal.

The program then enables the clockout signal during low power stop and activates the loss of clock monitor function. Next the program changes the VCO Clockout frequency to 32mhz, then enables the Clock failure monitor and enables the Reset on Clock or Lock failure functions. The program then enables the VCO Clockout signal.

The last step the program performs is to get the PLL runtime status. If an error occurs, the program exits indicating the line number where the failure occurred.

12.5.1.1 Example Application Code

```
/*********************************************************************

File:    pll_b_app.c

Purpose: The following example code illustrates the use of the M-CORE
         Peripherals Library PLL_B Level 1 driver.  It is setup to
         operation on an M-CORE 2107 CMB board.

         The program will first call the PLL_B_GetResetStatus_f to
         determines what options are available.

         The program will then initialize using the PLL_B_Init_f
         function with the frequency of 16mhz.

         Next the program will call PLL_B_ControlVCOOutput to disable
         the VCO Clockout signal.

         The program will then use the PLL_B_ControlStopMode to
         enable the clockout signal during low power stop and the
         PLL_B_ControlClockFailureResponse to activate the loss of
         clock monitor function.

         Next the program will call PLL_B_ControlVCOFrequency to
         change the VCO Clockout frequency to 32mhz.

         PLL_B_ControlClockFailureResponse will be called next to
         enable the Clock failure monitor and enable the Reset on
         Clock or Lock failure functions.

         The program will call PLL_B_ControlVCOOutput to enable
         the VCO Clockout signal.
```

```
        The last step will be to call PLL_B_GetRuntimeStatus.

        If an error occurs, the program will exit with the line
        number of where the failure occurred.

Time to run:
        About 10 seconds.

General Setup:
        General hardware and software setup to run this application
        can be found in the 'readme_app.txt' file, located in the
        '<library>/app' directory.

Application Specific Setup:
        Development Board: MMC2107 CMB or EVB

(C) Copyright Motorola Inc, 2000.  All rights reserved.
/*********************************************************************/

#include "pll_b.h"
#include <stdio.h>
#include "plibdefs_mmc2107.h"

/*********************************************************************/
/*  Macro Definitions                                                */
/*********************************************************************/
/* macro to handle error checking and response */
#define HANDLE_ERROR(err)   if (err != PLL_B_ERR_NONE) return(__LINE__);

#define PLL_B_WAIT_TIME 0x1000   /* Delay time to wait for phase lock */

int main(void);                         /* function prototype          */
int pll_b_example(pPLL_B_t);            /* function prototype          */
PLL_B_ReturnCode_t WaitforLockBit();

/*********************************************************************/
/* Name: main                                                        */
/* Description: Drive program operation                              */
/* Inputs: PLLPtr(pointer to PLL base address)                       */
/* Outputs: status (if passed, 0; else line number of the failure)   */
/*********************************************************************/
int main ()
{
    int status;

    status = pll_b_example ((pPLL_B_t)__MMC2107_PLL);

    return (status);
}

/*********************************************************************/
/* Name: pllb_example                                                */
/* Description: Provide an example for the usage of the PLL_B        */
```

```
/* Inputs:  PLLPtr (pointer to PLL_B base address)                    */
/* Outputs: status (if passed, 0; else line number of the failure)    */
/*********************************************************************/
int pll_b_example (pPLL_B_t PLLPtr)
{
   /*********************************************************************/
   /* declare local variables                                           */
   /*********************************************************************/

   PLL_B_ResetOnLockFailureControl_t   ResetOnLockFailureValue;
   PLL_B_VCOMultiplyFactorDivider_t    VCOMultiplicationFactorDivider;
   PLL_B_ResetOnClockFailureControl_t  ResetOnClockFailureValue;
   PLL_B_VCOReducedFrequencyDivider_t  VCOReducedFrequencyDivider;
   volatile
   PLL_B_MonitorClockFailure_t         MonitorClockFailureValue;
   PLL_B_DisableClockout_t             DisableClockoutValue;
   PLL_B_FastWakeupControl_t           FastWakeupValue;
   PLL_B_StopModeControl_t             StopModeValue;

   PLL_B_StatusAlert_t                 StatusAlertBlock;
   pPLL_B_StatusAlert_t                StatusAlertPtr
      = &StatusAlertBlock;

   PLL_B_Status_t                      StatusBlock;
   pPLL_B_Status_t                     StatusPtr
      = &StatusBlock;

   /* return codes */

   volatile PLL_B_ReturnCode_t status = PLL_B_ERR_NONE; /* return value        */
   volatile int status2 = PLL_B_ERR_NONE;               /* return value        */

   /*********************************************************************/
   /* Reset the PLL to the factory settings                             */
   /*********************************************************************/
   status =PLL_B_Reset
                   (
                      PLLPtr
                   );

   /*********************************************************************/
   /* Obtain the Reset Status by calling PLL_B_GetResetStatus           */
   /* [req.100.01]                                                      */
   /*********************************************************************/


   status = PLL_B_GetResetStatus
                            (
                               PLLPtr,
                               StatusPtr
                            );

    HANDLE_ERROR(status);
```

PLL_B Level 1

```
/*****************************************************************/
/* Initialize the PLL to 12mhz with PLL_B_Init  [req. 100.02]    */
/*****************************************************************/

ResetOnLockFailureValue        = PLL_B_RESET_ON_LOCK_FAILURE_DISABLE;
VCOMultiplicationFactorDivider = PLL_B_VCO_X_6;
ResetOnClockFailureValue       = PLL_B_RESET_ON_CLOCK_FAILURE_DISABLE;
VCOReducedFrequencyDivider     = PLL_B_VCO_DIV_4;
MonitorClockFailureValue       = PLL_B_MONITOR_CLOCK_FAILURE_DISABLE;
DisableClockoutValue           = PLL_B_CLOCK_OUT_ENABLE;
FastWakeupValue                = PLL_B_FAST_WAKEUP_DISABLE;
StopModeValue                  = PLL_B_DISABLE_ALL;

status = PLL_B_Init
                     (
                        PLLPtr,
                        ResetOnLockFailureValue,
                        VCOMultiplicationFactorDivider,
                        ResetOnClockFailureValue,
                        VCOReducedFrequencyDivider,
                        MonitorClockFailureValue,
                        DisableClockoutValue,
                        FastWakeupValue,
                        StopModeValue,
                        StatusAlertPtr,
                        WaitforLockBit
                     );

HANDLE_ERROR(status);

/*****************************************************************/
/* Setup actions to be take during low power stop mode by calling  */
/* PLL_B_ControlStopMode [req. 100.03]                           */
/*****************************************************************/

FastWakeupValue                = PLL_B_FAST_WAKEUP_ENABLE;
StopModeValue                  = PLL_B_ENABLE_PLL_OSC_CLKOUT;

status =  PLL_B_ControlStopMode
                     (
                        PLLPtr,
                        FastWakeupValue,
                        StopModeValue
                     );

HANDLE_ERROR(status);

/*****************************************************************/
/* Call PLL_B_ControlVCOOutput to disable the VCO Clockout signal  */
/* [req. 100.04]                                                 */
/*****************************************************************/

DisableClockoutValue           = PLL_B_CLOCK_OUT_DISABLE;
```

```
status = PLL_B_ControlVCOOutput
                    (
                       PLLPtr,
                       DisableClockoutValue
                    );

HANDLE_ERROR(status);

/*****************************************************************/
/* Call the VCO output Frequency to 32Mhz by calling             */
/* PLL_B_ControlVCOFrequency [req. 100.05]                       */
/*****************************************************************/

VCOMultiplicationFactorDivider = PLL_B_VCO_X_4;
VCOReducedFrequencyDivider     = PLL_B_VCO_DIV_1;

status2 =  PLL_B_ControlVCOFrequency
                    (
                       PLLPtr,
                       VCOMultiplicationFactorDivider,
                       VCOReducedFrequencyDivider,
                       StatusAlertPtr,
                       WaitforLockBit
                    );

HANDLE_ERROR(status2);

/*****************************************************************/
/* Setup actions to be taken in the event of clock or lock failure */
/* PLL_B_ControlClockFailureResponse [req. 100.06]                       */
/*****************************************************************/

MonitorClockFailureValue       = PLL_B_MONITOR_CLOCK_FAILURE_ENABLE;
ResetOnLockFailureValue        = PLL_B_RESET_ON_LOCK_FAILURE_ENABLE;
ResetOnClockFailureValue       = PLL_B_RESET_ON_CLOCK_FAILURE_ENABLE;

status = PLL_B_ControlClockFailureResponse
                    (
                       PLLPtr,
                       ResetOnLockFailureValue,
                       ResetOnClockFailureValue,
                       MonitorClockFailureValue
                    );

HANDLE_ERROR(status);

/*****************************************************************/
/* Call PLL_B_ControlVCOOutput to enable the VCO Clockout signal   */
/* [req. 100.07]                                                 */
/*****************************************************************/

DisableClockoutValue           = PLL_B_CLOCK_OUT_ENABLE;
```

PLL_B Level 1

```
   status = PLL_B_ControlVCOOutput
                        (
                           PLLPtr,
                           DisableClockoutValue
                        );

   HANDLE_ERROR(status);

   /*********************************************************************/
   /* Call PLL_B_GetRuntimeStatus to return the PLL runtime Status    */
   /* [req. 100.08]                                                     */
   /*********************************************************************/

   status = PLL_B_GetRuntimeStatus
                        (
                           PLLPtr,
                           StatusPtr
                        );

   HANDLE_ERROR(status);

return(status);
}


/*********************************************************************/
/* Function: WaitForLockBit                                          */
/* Purpose:  Optional wait function for the Phase Locking function.  */
/*********************************************************************/

PLL_B_ReturnCode_t WaitforLockBit()
{
   UINT32 i;

   for (i = 0; i < PLL_B_WAIT_TIME; i++)
   {
   }

  return PLL_B_ERR_NONE;
}
```

Section 13 Port_A Level 1

This section describes the Port_A Level 1 M•CORE™ Device Driver.

13.1 Overview

The MMC2107 Ports and their accompanying driver (Port_A) are used on the MMC2107 development system. The level 1 service is designed to allow the programmer to control the MMC2107 Ports on a near-register level, but without the concern of exact positions of registers and fields, or the detailed sequences of register operations. Refer to the MMC2107 Advance Information (Data Book) (MMC2107/D) for additional information.

13.2 Port_A Service Functions

The Port_A device driver software provides these functions:

- Resets all Port registers to their power-on reset state.
- Sets the data direction for a Port Pin.
- Writes to a Port Register.
- Gets Port pin states.
- Configures each of the Port Pins for the Port Pin Assignment Registers.
- Writes an 8-bit value to one of the Port Registers.
- Gets the contents of one of the Port Registers.
- Sets or clears a specified Port pin.

13.3 Port_A Data Type Definitions

The following paragraphs describe the data definitions recognized by the Port_A device driver. These data definitions are grouped into six categories: standard data types, board specific base address definitions, derived data types, enumerated data types, register macros, and return codes.

13.3.1 Standard Data Type Definitions

- UINT8 — 8-bit unsigned data value
- INT8 — 8-bit signed data value
- UINT16 — 16-bit unsigned data value
- INT16 — 16-bit signed data value
- UINT32 — 32-bit unsigned data value
- INT32 — 32-bit signed data value

13.3.2 Board Specific Base Address Definitions

```
#define __MMC2107_PORTS   (0x000C0000)
```

13.3.3 Derived Data Type Definitions

13.3.3.1 Port_A_t - Ports Register Definition

```
typedef struct {
   volatile UINT8   PORTA;
   volatile UINT8   PORTB;
   volatile UINT8   PORTC;
   volatile UINT8   PORTD;
   volatile UINT8   PORTE;
   volatile UINT8   PORTF;
   volatile UINT8   PORTG;
   volatile UINT8   PORTH;
   volatile UINT8   PORTI;
   volatile UINT8   RESERVED0;
   volatile UINT8   RESERVED1;
   volatile UINT8   RESERVED2;
            UINT8   DDRA;
            UINT8   DDRB;
            UINT8   DDRC;
            UINT8   DDRD;
            UINT8   DDRE;
            UINT8   DDRF;
            UINT8   DDRG;
            UINT8   DDRH;
            UINT8   DDRI;
            UINT8   RESERVED3;
            UINT8   RESERVED4;
            UINT8   RESERVED5;
   volatile UINT8   PORTAP_SETA;
   volatile UINT8   PORTBP_SETB;
   volatile UINT8   PORTCP_SETC;
   volatile UINT8   PORTDP_SETD;
   volatile UINT8   PORTEP_SETE;
   volatile UINT8   PORTFP_SETF;
   volatile UINT8   PORTGP_SETG;
   volatile UINT8   PORTHP_SETH;
   volatile UINT8   PORTIP_SETI;
            UINT8   RESERVED6;
            UINT8   RESERVED7;
            UINT8   RESERVED8;
            UINT8   CLRA;
            UINT8   CLRB;
            UINT8   CLRC;
            UINT8   CLRD;
            UINT8   CLRE;
            UINT8   CLRF;
            UINT8   CLRG;
```

```
            UINT8   CLRH;
            UINT8   CLRI;
            UINT8   RESERVED9;
            UINT8   RESERVED10;
            UINT8   RESERVED11;
            UINT8   PCDPAR;
            UINT8   PEPAR;
} Port_A_t, *pPort_A_t;
```

13.3.4 Enumerated Data Type Definitions

13.3.4.1 Port_A_Register_t - Ports Register Selection

```
typedef enum {
     PORT_A_PORTA,
     PORT_A_PORTB,
     PORT_A_PORTC,
     PORT_A_PORTD,
     PORT_A_PORTE,
     PORT_A_PORTF,
     PORT_A_PORTG,
     PORT_A_PORTH,
     PORT_A_PORTI,
     PORT_A_DDRA,
     PORT_A_DDRB,
     PORT_A_DDRC,
     PORT_A_DDRD,
     PORT_A_DDRE,
     PORT_A_DDRF,
     PORT_A_DDRG,
     PORT_A_DDRH,
     PORT_A_DDRI,
     PORT_A_PORTAP_SETA,
     PORT_A_PORTBP_SETB,
     PORT_A_PORTCP_SETC,
     PORT_A_PORTDP_SETD,
     PORT_A_PORTEP_SETE,
     PORT_A_PORTFP_SETF,
     PORT_A_PORTGP_SETG,
     PORT_A_PORTHP_SETH,
     PORT_A_PORTIP_SETI,
     PORT_A_CLRA,
     PORT_A_CLRB,
     PORT_A_CLRC,
     PORT_A_CLRD,
     PORT_A_CLRE,
     PORT_A_CLRF,
     PORT_A_CLRG,
     PORT_A_CLRH,
     PORT_A_CLRI,
     PORT_A_PCDPAR,
     PORT_A_PEPAR
} Port_A_Register_t;
```

13.3.4.2 Port_A_Function_t - Ports Mode Selection

```
typedef enum {
    PORT_A_DIGITAL_IO,  /* Register will be configured for digital input/output */
    PORT_A_PRIMARY_FUNCTION          /*Register will perform primary functions. */
} Port_A_Function_t;
```

13.3.4.3 Port_A__Selection_t – PEPAR Pin Selections

```
typedef enum {
    PORT_A_SHS,         /* Pin 7 of PEPAR register.     */
    PORT_A_TA,          /* Pin 6 of PEPAR register.     */
    PORT_A_TEA,         /* Pin 5 of PEPAR register.     */
    PORT_A_CSE,         /* Pin [4:3] of PEPAR register. */
    PORT_A_TC,          /* Pin [2:0] of PEPAR register. */
    PORT_A_Port_CD      /* Pin 7 of PCDPAR register.    */
} Port_A_Selection_t;
```

13.3.4.4 Port_A_Port_t – Port (A – I) Selections

```
typedef enum {
     PORT_A_PORT_A,         /* Port A selection. */
     PORT_A_PORT_B,         /* Port B selection. */
     PORT_A_PORT_C,         /* Port C selection. */
     PORT_A_PORT_D,         /* Port D selection. */
     PORT_A_PORT_E,         /* Port E selection. */
     PORT_A_PORT_F,         /* Port F selection. */
     PORT_A_PORT_G,         /* Port G selection. */
     PORT_A_PORT_H,         /* Port H selection. */
     PORT_A_PORT_I          /* Port I selection. */
} Port_A_Port_t;
```

13.3.4.5 Port_A_Pin_t - Port Pin Selection

```
typedef enum {
      PORT_A_PORTPIN_0,           /* Select Port Pin 0 */
      PORT_A_PORTPIN_1,           /* Select Port Pin 1 */
      PORT_A_PORTPIN_2,           /* Select Port Pin 2 */
      PORT_A_PORTPIN_3,           /* Select Port Pin 3 */
      PORT_A_PORTPIN_4,           /* Select Port Pin 4 */
      PORT_A_PORTPIN_5,           /* Select Port Pin 5 */
      PORT_A_PORTPIN_6,           /* Select Port Pin 6 */
      PORT_A_PORTPIN_7            /* Select Port Pin 7 */
} Port_A_Pin_t;
```

13.3.4.6 Port_A_DataVal_t – Port Pin Write Value

```
typedef enum {
      PORT_A_CLEAR,       /* Data Value will be cleared with a zero. */
      PORT_A_SET          /* Data Value will be set with a one.      */
} Port_A_DataVal_t;
```

13.3.4.7 Port_A_ReturnCode_t – PORT_A Error Codes

```
typedef enum {
    PORT_A_ERR_NONE,               /* no error                                  */
    PORT_A_ERR_INVALID_HANDLE,     /* Ports base address parameter is zero.     */
    PORT_A_ERR_INVALID_PIN_NUMBER, /* Pin selection is invalid.                 */
    PORT_A_ERR_INVALID_PORT,       /* Port selection is invalid.                */
    PORT_A_ERR_INVALID_DATA_VALUE, /* Value written into pin is not zero or one.*/
    PORT_A_ERR_BAD_RESULT_ADDRESS, /* Result pointer is zero.                   */
    PORT_A_ERR_INVALID_REGISTER,   /* Port Register selection is invalid.       */
    PORT_A_ERR_INVALID_FUNCTION,   /* Port Function selection is invalid.       */
    PORT_A_ERR_INVALID_SELECTION   /* Selection for pins is invalid.            */
} Port_A_ReturnCode_t;
```

13.3.5 Register Macros

13.3.5.1 Port Output Data Register Macros

```
#define PORTx_PORTx0_BITNO 0
#define PORTx_PORTx0_MASK  (1<<PORTx_PORTx0_BITNO)  /*0000 0001 */
#define PORTx_PORTx1_BITNO 1
#define PORTx_PORTx1_MASK  (1<<PORTx_PORTx1_BITNO)  /*0000 0010 */
#define PORTx_PORTx2_BITNO 2
#define PORTx_PORTx2_MASK  (1<<PORTx_PORTx2_BITNO)  /*0000 0100 */
#define PORTx_PORTx3_BITNO 3
#define PORTx_PORTx3_MASK  (1<<PORTx_PORTx3_BITNO)  /*0000 1000 */
#define PORTx_PORTx4_BITNO 4
#define PORTx_PORTx4_MASK  (1<<PORTx_PORTx4_BITNO)  /*0001 0000 */
#define PORTx_PORTx5_BITNO 5
#define PORTx_PORTx5_MASK  (1<<PORTx_PORTx5_BITNO)  /*0010 0000 */
#define PORTx_PORTx6_BITNO 6
#define PORTx_PORTx6_MASK  (1<<PORTx_PORTx6_BITNO)  /*0100 0000 */
#define PORTx_PORTx7_BITNO 7
#define PORTx_PORTx7_MASK  (1<<PORTx_PORTx7_BITNO)  /*1000 0000 */
#define PORTx_PORTx_BITNO 7
#define PORTx_PORTx_MAX 0xFF
#define PORTx_PORTx_MASK (PORTx_PORTx_MAX<<PORTx_PORTx_BITNO)
#define PORTx_RESET_MASK (0xFF)
```

13.3.5.2 Port Data Direction Register

```
#define DDRx_DDRx0_BITNO 0
#define DDRx_DDRx0_MASK  (1<<DDRx_DDRx0_BITNO)        /*0000 0001 */
#define DDRx_DDRx1_BITNO 1
#define DDRx_DDRx1_MASK  (1<<DDRx_DDRx1_BITNO)        /*0000 0010 */
#define DDRx_DDRx2_BITNO 2
#define DDRx_DDRx2_MASK  (1<<DDRx_DDRx2_BITNO)        /*0000 0100 */
#define DDRx_DDRx3_BITNO 3
#define DDRx_DDRx3_MASK  (1<<DDRx_DDRx3_BITNO)        /*0000 1000 */
#define DDRx_DDRx4_BITNO 4
#define DDRx_DDRx4_MASK  (1<<DDRx_DDRx4_BITNO)        /*0001 0000 */
```

Port_A Level 1

```
#define DDRx_DDRx5_BITNO 5
#define DDRx_DDRx5_MASK  (1<<DDRx_DDRx5_BITNO)        /*0010 0000 */
#define DDRx_DDRx6_BITNO 6
#define DDRx_DDRx6_MASK  (1<<DDRx_DDRx6_BITNO)        /*0100 0000 */
#define DDRx_DDRx7_BITNO 7
#define DDRx_DDRx7_MASK  (1<<DDRx_DDRx7_BITNO)        /*1000 0000 */
#define DDRx_DDRx_BITNO  7
#define DDRx_DDRx_MAX 0XFF
#define DDRx_DDRx_MASK  (DDRx_DDRx_MAX<<DDRx_DDRx_BITNO)
#define DDRx_RESET_MASK (0x00)
```

13.3.5.3 Port Pin/Set Data Registers

```
#define PORTxP_SETx_PORTxP0_SETx0_BITNO  0
#define PORTxP_SETx_PORTxP0_SETx0_MASK    (1<<PORTxP_SETx_PORTxP0_SETx0_BITNO)
#define PORTxP_SETx_PORTxP1_SETx1_BITNO  1
#define PORTxP_SETx_PORTxP1_SETx1_MASK    (1<< PORTxP_SETx_PORTxP1_SETx1_BITNO)
#define PORTxP_SETx_PORTxP2_SETx2_BITNO  2
#define PORTxP_SETx_PORTxP2_SETx2_MASK    (1<<PORTxP_SETx_PORTxP2_SETx2_BITNO)
#define PORTxP_SETx_PORTxP3_SETx3_BITNO  3
#define PORTxP_SETx_PORTxP3_SETx3_MASK    (1<< PORTxP_SETx_PORTxP3_SETx3_BITNO)
#define PORTxP_SETx_PORTxP4_SETx4_BITNO  4
#define PORTxP_SETx_PORTxP4_SETx4_MASK    (1<<PORTxP_SETx_PORTxP4_SETx4_BITNO)
#define PORTxP_SETx_PORTxP5_SETx5_BITNO  5
#define PORTxP_SETx_PORTxP5_SETx5_MASK    (1<< PORTxP_SETx_PORTxP5_SETx5_BITNO)
#define PORTxP_SETx_PORTxP6_SETx6_BITNO  6
#define PORTxP_SETx_PORTxP6_SETx6_MASK    (1<<PORTxP_SETx_PORTxP6_SETx6_BITNO)
#define PORTxP_SETx_PORTxP7_SETx7_BITNO  7
#define PORTxP_SETx_PORTxP7_SETx7_MASK    (1<<PORTxP_SETx_PORTxP7_SETx7_BITNO)
#define PORTxP_SETx_PORTxP_SETx_BITNO    7
#define PORTxP_SETx_PORTxP_SETx_MAX    0xFF
#define PORTxP_SETx_PORTxP_SETx_MASK (PORTxP_SETx_PORTxP_SETx_MAX<<
PORTxP_SETx_PORTxP_SETx_BITNO)
```

13.3.5.4 Port Clear Output Data Registers

```
#define CLRx_CLRx0_BITNO 0
#define CLRx_CLRx0_MASK  (1<<CLR_CLRx0_BITNO)
#define CLRx_CLRx1_BITNO 1
#define CLRx_CLRx1_MASK  (1<<CLR_CLRx1_BITNO)
#define CLRx_CLRx2_BITNO 2
#define CLRx_CLRx2_MASK  (1<<CLR_CLRx2_BITNO)
#define CLRx_CLRx3_BITNO 3
#define CLRx_CLRx3_MASK  (1<<CLR_CLRx3_BITNO)
#define CLRx_CLRx4_BITNO 4
#define CLRx_CLRx4_MASK  (1<<CLR_CLRx4_BITNO)
#define CLRx_CLRx5_BITNO 5
#define CLRx_CLRx5_MASK  (1<<CLR_CLRx5_BITNO)
#define CLRx_CLRx6_BITNO 6
#define CLRx_CLRx6_MASK  (1<<CLR_CLRx6_BITNO)
#define CLRx_CLRx7_BITNO 7
#define CLRx_CLRx7_MASK  (1<<CLR_CLRx7_BITNO)
#define CLRx_CLRx_BITNO 7
#define CLRx_CLRx_MAX  (0XFF)
#define CLRx_CLRx_MASK  (CLRx_CLRx_MAX<< CLRx_CLRx_BITNO)
#define CLRx_RESET_MASK  (0x00)
```

13.3.5.5 Port Pin Assignment Registers

```
#define PCDPAR_PCDPA_BITNO 7
#define PCDPAR_PCDPA_MASK  (1<<PCDPAR_PCDPA_BITNO)
#define PCDPAR_RESET_MASTER_MASK  (0x80)
#define PCDPAR_RESET_SINGLE_MASK  (0x00)
#define PCDPAR_RESET_EMULATION_MASK (0x00)

#define PEPAR_PEPA0_BITNO 0
#define PEPAR_PEPA0_MASK   (1<<PEPAR_PEPA0_BITNO)
#define PEPAR_PEPA1_BITNO 1
#define PEPAR_PEPA1_MASK   (1<<PEPAR_PEPA1_BITNO)
#define PEPAR_PEPA2_BITNO 2
#define PEPAR_PEPA2_MASK   (1<<PEPAR_PEPA2_BITNO)
#define PEPAR_PEPA3_BITNO 3
#define PEPAR_PEPA3_MASK   (1<<PEPAR_PEPA3_BITNO)
#define PEPAR_PEPA4_BITNO 4
#define PEPAR_PEPA4_MASK   (1<<PEPAR_PEPA4_BITNO)
#define PEPAR_PEPA5_BITNO 5
#define PEPAR_PEPA5_MASK   (1<<PEPAR_PEPA5_BITNO)
#define PEPAR_PEPA6_BITNO 6
#define PEPAR_PEPA6_MASK   (1<<PEPAR_PEPA6_BITNO)
#define PEPAR_PEPA7_BITNO 7
#define PEPAR_PEPA7_MASK   (1<<PEPAR_PEPA7_BITNO)
#define PEPAR_PEPA_BITNO  0
#define PEPAR_PEPA _MASK   (0XFF)
#define PEPAR_RESET_MASTER_MASK (0XE0)
#define PEPAR_RESET_SINGLE_MASK (0X00)
#define PEPAR_RESET_EMULATION_MASK (0XFF)
```

13.3.6 Return Codes

Return codes from the enumerated type “Port_A_ReturnCode_t”.

Return Code	Functions Returning	Description
PORT_A_ERR_NONE	All	No error.
PORT_A_ERR_INVALID_HANDLE	All	Ports base address parameter is zero.
PORT_A_ERR_INVALID_PIN_NUMBER	ConfigureGPIO	Pin Selection is invalid.
PORT_A_ERR_INVALID_PORT	ConfigureDataDirection, WritePortData, ReadPortData	The port selection is invalid.
PORT_A_ERR_INVALID_DATA_VALUE	WritePin	Value written into pin is not zero or one.
PORT_A_ERR_BAD_RESULT_ADDRESS	ReadPortData, GetRegister	Result Pointer is zero.
PORT_A_ERR_INVALID_REGISTER	SetRegister, GetRegister	Port Register selection is invalid.
PORT_A_ERR_INVALID_FUNCTION	ConfigureGPIO	Port Function Selection is invalid.
PORT_A_ERR_INVALID_SELECTION	ConfigureGPIO	The selection chosen is invalid.

13.4 Port_A API Definitions

13.4.1 PORT_A_Reset

13.4.1.1 Description

The Port_A_Reset function resets all Port registers to their power-on reset state. Parameter checking is a compile-time option.

This function can be called at any time.

13.4.1.2 Prototype

```
Port_A_ReturnCode_t Port_A_Reset (
          pPort_A_t    PortPtr
)
```

13.4.1.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPort_A_t	PortPtr	PORT_A base address associated with this driver.	Any non-zero core processor data address.

13.4.1.4 Return Values

Error Code	Description
PORT_A_ERR_NONE	No error
PORT_A_ERR_INVALID_HANDLE	Ports base address parameter is zero.

13.4.2 PORT_A_ConfigureDataDirection

13.4.2.1 Description

The Port_A_ConfigureDataDirection function sets the data direction for a Port Pin. Parameter checking is a compile-time option.

This function can be called at any time if used as General Purpose I/O otherwise it should be called after the Port has been reset through a previous call to the PORT_A_Reset function.

13.4.2.2 Prototype

```
Port_A_ReturnCode_t Port_A_ConfigureDataDirection (
     pPort_A_t        PortPtr,
     Port_A_Port_t    Port,
     UINT8            DataDirMask
)
```

Port_A Level 1

13.4.2.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPort_A_t	PortPtr	Port base address associated with this driver.	Any non-zero core processor data address.
Port_A_Port_t	Port	Port (A – I) Selection	PORT_A_PORT_A, PORT_A_PORT_B PORT_A_PORT_C, PORT_A_PORT_D, PORT_A_PORT_E, PORT_A_PORT_F, PORT_A_PORT_G, PORT_A_PORT_H, PORT_A_PORT_I
UINT8	DataDirMask	Data to copy to register.	Any 8-bit value.

13.4.2.4 Return Values

Error Code	Description
PORT_A_ERR_NONE	No error
PORT_A_ERR_INVALID_HANDLE	PORT base address parameter is zero
PORT_A_ERR_INVALID_PORT	Port selection is invalid.

13.4.3 PORT_A_WritePortData

13.4.3.1 Description

The Port_A_WritePortData function writes unformatted data to a Port register. Parameter checking is a compile-time option.

This function can be called at any time.

13.4.3.2 Prototype

```
Port_A_ReturnCode_t Port_A_WritePortData (
      pPort_A_t       PortPtr,
      Port_A_Port_t   Port,
      UINT8           WriteMask
)
```

13.4.3.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPort_A_t	PortPtr	Port base address associated with this driver.	Any non-zero core processor data address.
Port_A_Port_t	Port	Port (A – I) Selection	PORT_A_PORT_A, PORT_A_PORT_B PORT_A_PORT_C, PORT_A_PORT_D, PORT_A_PORT_E, PORT_A_PORT_F, PORT_A_PORT_G, PORT_A_PORT_H, PORT_A_PORT_I
UINT8	WriteMask	Data to write to register.	Any 8-bit value.

13.4.3.4 Return Values

Error Code	Description
PORT_A_ERR_NONE	No error
PORT_A_ERR_INVALID_HANDLE	Port base address parameter is zero
PORT_A_ERR_INVALID_PORT	Port selection is invalid.

13.4.4 PORT_A_ReadPortData

13.4.4.1 Description

The Port_A_ReadPortData function returns Port pin states. Parameter checking is a compile time option.

This function can be called at any time.

13.4.4.2 Prototype

```
Port_A_ReturnCode_t Port_A_ ReadPortData (
          pPort_A_t        PortPtr,
          Port_A_Port_t    Port,
          UINT8           *ReadPortPtr
)
```

Port_A Level 1

13.4.4.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPort_A_t	PortPtr	Port base address associated with this driver.	Any non-zero core processor data address.
Port_A_Port_t	Port	Port (A – I) Selection	PORT_A_PORT_A, PORT_A_PORT_B PORT_A_PORT_C, PORT_A_PORT_D, PORT_A_PORT_E, PORT_A_PORT_F, PORT_A_PORT_G, PORT_A_PORT_H, PORT_A_PORT_I
UINT8	*ReadPortPtr	Return Address for received data.	Any non-zero core processor data address.

13.4.4.4 Return Values

Error Code	Description
PORT_A_ERR_NONE	No error
PORT_A_ERR_INVALID_HANDLE	Port base address parameter is zero
PORT_A_ERR_BAD_RESULT_ADDRESS	Result Pointer is zero.
PORT_A_ERR_INVALID_PORT	Port selection is invalid.

13.4.5 PORT_A_ConfigureGPIO

13.4.5.1 Description

The Port_A_ConfigureGPIO function configures each of the Port Pins for the Port Pin Assignment Registers. Parameter checking is a compile-time option.

This function can be called at any time.

13.4.5.2 Prototype

```
Port_A_ReturnCode_t Port_A_ConfigureGPIO (
            pPort_A_t             PortPtr,
            Port_A_Selection_t    SelectValue,
            Port_A_Function_t     FunctionType
)
```

13.4.5.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPort_A_t	PortPtr	Port base address associated with this driver.	Any non-zero core processor data address.
Port_A_Selection_t	SelectValue	Select pin assignment.	PORT_A_SHS, PORT_A_TA, PORT_A_TEA, PORT_A_CSE, PORT_A_TC, PORT_A_Port_CD
Port_A_Function_t	FunctionType	Value to write to the Port Pin.	PORT_A_DIGITAL_IO, PORT_A_PRIMARY_FUNCTION

13.4.5.4 Return Values

Error Code	Description
PORT_A_ERR_NONE	No error
PORT_A_ERR_INVALID_HANDLE	Port base address parameter is zero.
PORT_A_ERR_INVALID_SELECTION	Selection chosen is invalid.
PORT_A_ERR_INVALID_FUNCTION	Function selection is invalid.

13.4.6 PORT_A_SetRegister

13.4.6.1 Description

The Port_A_SetRegister function writes unformatted data to a specified Port register. Parameter checking is a compile-time option.

This function can be called at any time.

13.4.6.2 Prototype

```
Port_A_ReturnCode_t Port_A_SetRegister (
        pPort_A_t           PortPtr,
        Port_A_Register_t   PortRegister,
        UINT8               RegisterValue
)
```

Port_A Level 1

13.4.6.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPort_A_t	PortPtr	Port base address associated with this driver.	Any non-zero core processor data address.
Port_A_Register_t	PortRegister	Select among PORT_A registers.	Any enumerations of Port_A_Register_t.
UINT8	RegisterValue	Data to copy into selected register.	Any 8-bit value.

13.4.6.4 Return Values

Error Code	Description
PORT_A_ERR_NONE	No error
PORT_A_ERR_INVALID_HANDLE	Port base address parameter is zero.
PORT_A_ERR_INVALID_REGISTER	Port Register selection is invalid.

13.4.7 PORT_A_GetRegister

13.4.7.1 Description

The PORT_A_GetRegister function returns the contents of one of the Port Registers. Parameter checking is a compile-time option.

This function can be called at any time.

13.4.7.2 Prototype

```
Port_A_ReturnCode_t Port_A_GetRegister (
          pPort_A_t            PortPtr,
          Port_A_Register_t    PortRegister,
          UINT8               *GetRegisterPtr
)
```

13.4.7.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPort_A_t	PortPtr	Port base address associated with this driver.	Any non-zero core processor data address.
Port_A_Register_t	PortRegister	Select among PORT_A registers.	Any enumeration of Port_A_Register_t
UINT8	*GetRegisterPtr	Result Address for Selected Port Register.	Any non-zero core processor data address.

13.4.7.4 Return Values

Error Code	Description
PORT_A_ERR_NONE	No error
PORT_A_ERR_INVALID_HANDLE	Port base address parameter is zero.
PORT_A_ERR_BAD_RESULT_ADDRESS	Result Pointer is zero.
PORT_A_ERR_INVALID_REGISTER	Port Register selection is invalid.

13.4.8 PORT_A_WritePin

13.4.8.1 Description

The Port_A_WritePin function sets or clears a specified Port pin. Parameter checking is a compile-time option.

The function can be called at any time.

13.4.8.2 Prototype

```
Port_A_ReturnCode_t Port_A_WritePin (
          pPort_A_t           PortPtr,
          Port_A_Port_t       Port,
          Port_A_Pin_t        PortPin,
          Port_A_DataVal_t    DataVal
)
```

Port_A Level 1

13.4.8.3 Arguments

Data Type	Formal Name	Description	Valid Values
pPort_A_t	PortPtr	Port base address associated with this driver.	Any non-zero core processor data address.
Port_A_Port_t	Port	Port (A – I) Selection	PORT_A_PORT_A, PORT_A_PORT_B, PORT_A_PORT_C, PORT_A_PORT_D, PORT_A_PORT_E, PORT_A_PORT_F, PORT_A_PORT_G, PORT_A_PORT_H, PORT_A_PORT_I
Port_A_Pin_t	PortPin	Selection of Port Pin.	PORT_A_PORTPIN_0, PORT_A_PORTPIN_1, PORT_A_PORTPIN_2, PORT_A_PORTPIN_3, PORT_A_PORTPIN_4, PORT_A_PORTPIN_5, PORT_A_PORTPIN_6, PORT_A_PORTPIN_7
Port_A_DataVal_t	DataVal	Value to write to port pin.	PORT_A_CLEAR, PORT_A_SET

13.4.8.4 Return Values

Error Code	Description
PORT_A_ERR_NONE	No error
PORT_A_ERR_INVALID_HANDLE	Port base address parameter is zero.
PORT_A_ERR_INVALID_PORT	Port selection is invalid.
PORT_A_ERR_INVALID_PIN_NUMBER	Pin Selected is invalid.
PORT_A_ERR_INVALID_DATA_VALUE	Value written into pin is not zero or one.

13.5 Port_A Example Application

The following describes the Port_A example application.

13.5.1 Description for Example Application

This example program resets all the Port registers to their power on state. The program configures pin DDRA4, DDRA3 and DDRA2 of the Port Data Direction Register as outputs. The program will then configure DDRA7, DDRA6 and DDRA5 as inputs. If a high signal is received as input, the LED on the MMC2107 board will turn on. If a low signal is received as input, the LED on the MMC2107 board will remain off. Another way that the data that is inside the register can be read is by tying the input pins to the output pins.

13.5.1.1 Example Application Code

```
/*********************************************************************

File:    porta_app.c

Purpose: The following example code illustrates the use of the M-CORE
         Peripherals Library Port_A Level 1 driver.  It is setup
         to operate on an M-CORE 2107 CMB board.

         This program can only run in single chip mode.  To configure
         the board to run in this mode to switches have to be turned
         on or off.  Switch name M1 and M0 have to be ON and OFF
         respectively for the board to boot in single chip mode.
         All the User Switches and LEDs in this application are
         controlled by Port H.  The LEDs are controlled by Port H
         bits 3....0 respectively.  To enable LED control and Sw
         access Port H bit 7 must be programmed as an output and be set
         to low.  To accomplish this a 8-bit mask will be supplied
         that will configure the pins as input or output.  The input
         pins will be 6,5 and 4.  The output pins will be 7,3,2,1 and
         0.  The program will reset all the bits and then configure
         the pins as aforementioned.  The program will then read
         the pins to verify that data that was inputted is correct.

         This program will continue running until all pins are
         configured or it has exceeded the counter value.

         If an error occurs, the program will exit with the line
         number of where the failure occurred.

         NOTE: This application can only run in single chip mode and
         must use the following files to do so:
         Memcfg_Single_Chip_2107.txt and Single_Chip_2107.cmd


Time to run:
         About 30 seconds.
```

Port_A Level 1

```
General Setup:
         General hardware and software setup to run this application
         can be found in the 'readme_app.txt' file, located in the
         '<library>/app' directory.

Application Specific Setup:
         Development Board: MMC2107 CMB or EVB

(C) Copyright Motorola Inc, 2000.  All rights reserved.
/*****************************************************************/

#include "port_a.h"
#include "plibdefs_mmc2107.h"

/*****************************************************************/
/*  Macro Definitions                                            */
/*****************************************************************/
/* macro to handle error checking and response */
#define HANDLE_ERROR(err) if (err != PORT_A_ERR_NONE) return(__LINE__);
#ifndef __MMC2107_PORTS
#define __MMC2107_PORTS ((pPort_A_t) (0x00C00000))
#endif

/*****************************************************************/
/*  Function Prototypes                                          */
/*****************************************************************/

int main(void);                          /* function prototype */
int porta_example(pPort_A_t);     /* function prototype */

/*****************************************************************/
/* Name: main                                                    */
/* Description: Drive program operation                          */
/* Inputs: PortPtr(pointer to Port_A base address)               */
/* Outputs: status (if passed, 0; else line number of the        */
/* failures)                                                     */
/*****************************************************************/
int main ()
{
    int status;

    status = porta_example ((pPort_A_t)__MMC2107_PORTS);
    return (status);
}

/*****************************************************************/
/* Name: porta_example                                           */
/* Description: Provide an example for the usage of the          */
/*              Port_A.                                          */
/* Inputs:  PortPtr (pointer to Port_A base address)             */
/* Outputs: status (if passed, 0; else line number of the        */
/*          failure)                                             */
/*****************************************************************/
```

```
int porta_example (pPort_A_t PortPtr)
{
/*---------------------------------------------------------------*/
/* declare local variables                                       */
/*---------------------------------------------------------------*/
/* return codes */
Port_A_ReturnCode_t status=PORT_A_ERR_NONE; /*return value*/
UINT8 DataDirMask=0x8F;
UINT8  ReadValue = 0;
UINT8 *ReadPortPtr=&ReadValue;

/* EdgePort_B_Port_t EdgePortPin;*/

/*---------------------------------------------------------------*/
/* resets the  Port                                              */
/*---------------------------------------------------------------*/
/* Call Port_A_Reset with valid parameters */
status = Port_A_Reset(PortPtr);

/* If an error code is returned, exit the program */
HANDLE_ERROR(status);

/*---------------------------------------------------------------*/
/* Configure Port H                                              */
/*---------------------------------------------------------------*/


status = Port_A_ConfigureDataDirection(
      PortPtr,                        /* Edge Port base address  */
      PORT_A_PORT_H,                   /* Port desired.          */
      DataDirMask                        /* Data Direction Mask  */
                                    );

/*---------------------------------------------------------------*/
/* write clear value of 1 into Pin 3 of Port H                   */
/*---------------------------------------------------------------*/


status = Port_A_WritePin(
      PortPtr,                       /* Edge Port base address  */
      PORT_A_PORT_H,                  /* Port desired.          */
      PORT_A_PORTPIN_3,
      PORT_A_SET
                               );
/*---------------------------------------------------------------*/
/* Read Value that is in address that ReadPortPtr Points to.     */
/*---------------------------------------------------------------*/

status = Port_A_ReadPortData(
      PortPtr,                       /* Edge Port base address  */
      PORT_A_PORT_H,                  /* Port desired.          */
      ReadPortPtr                 /* for storage of read value   */
                              );
   return(status);
```

Port_A Level 1

}

Section 14 QADC64_A Level 1

This section describes the QADC64_A Level 1 M•CORE™ Device Driver.

14.1 Overview

The Queued Analog-to-Digital Converter (QADC64) and its accompanying driver (QADC64_A) are used on the MMC2107 development system. The level 1 service is designed to allow the programmer to control the QADC64 on a near-register level, but without the concern of exact positions of registers and fields, or the detailed sequences of register operations. Refer to the MMC2107 Advance Information (Data Book) (MMC2107/D) for additional information.

14.2 QADC64_A Service Functions

The QADC64_A device driver software provides these functions:

- Writes a user-supplied data value to a user-specified QADC Register
- Gets data from a QADC register
- Initializes the QADC
- Initializes the operation of a QADC Queue
- Sets a QADC Conversion Command Word (CCW)
- Gets the status of the QADC
- Enables or disables QADC conversion
- Resets a QADC status flags
- Reads an entry in the QADC Result Word Table
- Sets a QADC Port A Pin
- Gets a QADC Port Pin

14.3 QADC64_A Data Type Definitions

The following paragraphs describe the data definitions recognized by the QADC64_A device driver. These data definitions are grouped into six categories: standard data types, board specific base address definitions, abstract data types, enumerated data types, register macros, and return codes.

14.3.1 Standard Data Type Definitions

- UINT8 — 8-bit unsigned data value
- UINT16 — 16-bit unsigned data value
- QADC64_A_Boolean_t — QADC64_A_Boolean_t

14.3.2 Board Specific Base Address Definitions

```
#define __MMC2107_QADC 0x00CA0000
```

14.3.3 Abstract Data Type Definitions

14.3.3.1 QADC64_A_t – QADC64 Register Definition

```
typedef struct {
               UINT16 QADCMCR;        /* Module Configuration Register       */
               UINT16 QADCTEST;       /* Test Register                       */
               UINT16 QADCINT;        /* Interrupt Register                  */
     volatile  UINT8  PORTQA;         /* Port A Data                         */
     volatile  UINT8  PORTQB;         /* Port B Data                         */
               UINT16 DDRQA;          /* Port Data Direction Register        */
               UINT16 QACR0;          /* Control Register 0                  */
     volatile  UINT16 QACR1;          /* Control Register 1                  */
     volatile  UINT16 QACR2;          /* Control Register 2                  */
     volatile  UINT16 QASR0;          /* Status Register 0                   */
     volatile  UINT16 QASR1;          /* Status Register 1                   */
               UINT16 RESERVED[245];  /* RESERVED $014-1FE                   */
               UINT16 CCW[64];        /* Conversion Command Word Table       */
     volatile  UINT16 RJURR[64];      /* Result Word Table, Right Justified, */
                                      /* Unsigned Result Register            */
     volatile  UINT16 LJSRR[64];      /* Result Word Table, Left Justified,  */
                                      /* Signed Result Register              */
     volatile  UINT16 LJURR[64];      /* Result Word Table, Left Justified,  */
                                      /* Unsigned Result Register            */
} QADC64_A_t, *pQADC64_A_t;
```

14.3.3.2 QADC64_A_Status_t – QADC64 Status Definition

```
typedef struct {
          QADC64_A_Boolean_t    CompletionFlag;      /* QASR0-CFx       */
          QADC64_A_Boolean_t    PauseFlag;           /* QASR0-PFx       */
          QADC64_A_Boolean_t    TriggerOverrunFlag;  /* QASR0-TORx      */
          UINT8                 QueueStatus;         /* QASR0-QS field  */
          UINT8                 CommandWordPointer;  /* QASR0-CWP       */
          UINT8                 QueueCWP;            /* QASR1-CWPQx     */
} QADC64_A_Status_t, *pQADC64_A_Status_t;
```

14.3.4 Enumerated Data Type Definitions

14.3.4.1 QADC64_A_RegisterSwitch_t - QADC64_A Register Selection

```
typedef enum {                  /* Get and set register selections */
     QADC64_A_QADCMCR_SWITCH,   /* Select QADCMCR       */
     QADC64_A_QADCINT_SWITCH,   /* Select QADCINT       */
     QADC64_A_PORTQA_SWITCH,    /* Select PORTQA        */
     QADC64_A_PORTQB_SWITCH,    /* Select PORTQB        */
     QADC64_A_DDRQA_SWITCH,     /* Select DDRQA         */
     QADC64_A_QACR0_SWITCH,     /* Select QACR0         */
     QADC64_A_QACR1_SWITCH,     /* Select QACR1         */
     QADC64_A_QACR2_SWITCH,     /* Select QACR2         */
     QADC64_A_QASR0_SWITCH,     /* Select QASR0         */
     QADC64_A_QASR1_SWITCH      /* Select QASR1         */
} QADC64_A_RegisterSwitch_t;
```

14.3.4.2 QADC64_A_DataFormat_t - QADC64_A Data Format Selection

```
typedef enum {      /* Result Word Data Format selections */
     QADC64_A_RJURR_DATA_FORMAT, /* Right Justified, Unsigned Result Register*/
     QADC64_A_LJSRR_DATA_FORMAT, /* Left Justified, Signed Result Register   */
     QADC64_A_LJURR_DATA_FORMAT  /* Left Justified, Unsigned Result Register */
} QADC64_A_DataFormat_t;
```

14.3.4.3 QADC64_A_TriggerAssignment_t - QADC64_A External Trigger Assignment

```
typedef enum {       /* Trigger/Queue Assignment                  */
      ETRIG1_QUEUE1, /* ETRIG1/Queue1, ETRIG2/Queue2 Assignment   */
      ETRIG1_QUEUE2, /* ETRIG1/Queue2, ETRIG2/Queue1 Assignment   */
} QADC64_A_TriggerAssignment_t;
```

14.3.4.4 QADC64_A_ReturnCode_t – QADC64_A Error Return Codes

```
typedef enum {
   QADC64_A_ERR_NONE = 0x0L,                /* no error                          */
   QADC64_A_ERR_INVALID_HANDLE,             /* PLL base address is zero          */
   QADC64_A_ERR_BAD_RESULT_ADDR,            /* Result pointer is zero            */
   QADC64_A_ERR_INVALID_REGISTER,           /* Register selection if invalid     */
   QADC64_A_ERR_INVALID_DATA_FORMAT,        /* Data Format is invalid            */
   QADC64_A_ERR_INVALID_IARB,               /* Aribitration flag – no applicable */
   QADC64_A_ERR_INVALID_QCLK_PSH,           /* High Time QCLK Prescaler Clck     */
   QADC64_A_ERR_INVALID_QCLK_PSL,           /* Low Time QCLK Prescaler Clock     */
   QADC64_A_ERR_INVALID_TRIGGER_ASSIGNMENT, /* Invalid Trigger Assignment        */
   QADC64_A_ERR_INVALID_IRL,                /* Invalid Interrupt Request         */
   QADC64_A_ERR_INVALID_BQ2,                /* Invalid Beginning of Queue 2      */
   QADC64_A_ERR_INVALID_CCW,                /* Invalid Conversion Command        */
   QADC64_A_ERR_INVALID_IST,                /* Invalid Input Sample Time         */
   QADC64_A_ERR_INVALID_CHANNEL,            /* Invalid Channel Number            */
   QADC64_A_ERR_INVALID_MODE,               /* Specified Queue Operating         */
   QADC64_A_ERR_INVALID_RESET,              /* Specified Status Reset Flags      */
   QADC64_A_ERR_INVALID_RESULT_WORD         /* Table entry is invalid            */
} QADC64_A_ReturnCode_t;
```

14.3.4.5 QADC64_A_Boolean_t - QADC64_A Boolean Data Type

```
typedef enum {
   QADC64_A_FALSE,          /* QADC false flag indicator      */
   QADC64_A_TRUE            /* QADC true flag indicator       */
} QADC64_A_Boolean_t;
```

14.3.5 Register Macros

14.3.5.1 PORTQA/PORTQB Register Pin Macros

```
#define PORTQA_PQA7_BITNO    15
#define PORTQA_PQA7_MASK     (1 << PORTQA_PQA7_BITNO)
#define PORTQA_PQA6_BITNO    14
#define PORTQA_PQA6_MASK     (1 << PORTQA_PQA6_BITNO)
#define PORTQA_PQA5_BITNO    13
#define PORTQA_PQA5_MASK     (1 << PORTQA_PQA5_BITNO)
#define PORTQA_PQA4_BITNO    12
#define PORTQA_PQA4_MASK     (1 << PORTQA_PQA4_BITNO)
#define PORTQA_PQA3_BITNO    11
#define PORTQA_PQA3_MASK     (1 << PORTQA_PQA3_BITNO)
#define PORTQA_PQA2_BITNO    10
#define PORTQA_PQA2_MASK     (1 << PORTQA_PQA2_BITNO)
#define PORTQA_PQA1_BITNO    9
#define PORTQA_PQA1_MASK     (1 << PORTQA_PQA1_BITNO)
#define PORTQA_PQA0_BITNO    8
#define PORTQA_PQA0_MASK     (1 << PORTQA_PQA0_BITNO)
#define PORTQB_PQB7_BITNO    7
#define PORTQB_PQB7_MASK     (1 << PORTQB_PQB7_BITNO)
#define PORTQB_PQB6_BITNO    6
```

```
#define PORTQB_PQB6_MASK     (1 << PORTQB_PQB6_BITNO)
#define PORTQB_PQB5_BITNO    5
#define PORTQB_PQB5_MASK     (1 << PORTQB_PQB5_BITNO)
#define PORTQB_PQB4_BITNO    4
#define PORTQB_PQB4_MASK     (1 << PORTQB_PQB4_BITNO)
#define PORTQB_PQB3_BITNO    3
#define PORTQB_PQB3_MASK     (1 << PORTQB_PQB3_BITNO)
#define PORTQB_PQB2_BITNO    2
#define PORTQB_PQB2_MASK     (1 << PORTQB_PQB2_BITNO)
#define PORTQB_PQB1_BITNO    1
#define PORTQB_PQB1_MASK     (1 << PORTQB_PQB1_BITNO)
#define PORTQB_PQB0_BITNO    0
#define PORTQB_PQB0_MASK     (1 << PORTQB_PQB0_BITNO)
```

14.3.5.2 QASR0 Status Reset Macros

```
/* These macros can be used with the QASR0 */
#define QASR0_CF1_BITNO    15
#define QASR0_CF1_MASK     (1 << QASR0_CF1_BITNO)
#define QASR0_PF1_BITNO    14
#define QASR0_PF1_MASK     (1 << QASR0_PF1_BITNO)
#define QASR0_CF2_BITNO    13
#define QASR0_CF2_MASK     (1 << QASR0_CF2_BITNO)
#define QASR0_PF2_BITNO    12
#define QASR0_PF2_MASK     (1 << QASR0_PF2_BITNO)
#define QASR0_TOR1_BITNO   11
#define QASR0_TOR1_MASK    (1 << QASR0_TOR1_BITNO)
#define QASR0_TOR2_BITNO   10
#define QASR0_TOR2_MASK    (1 << QASR0_TOR2_BITNO)
```

14.3.6 Return Codes

Return codes from the enumerated type “QADC64_A_ReturnCode_t”.

Return Code	Functions Returning	Description
QADC64_A_ERR_NONE	All	No error.
QADC64_A_ERR_INVALID_HANDLE	All	QSMCM base address parameter is zero
QADC64_A_ERR_INVALID_REGISTER	QADC64_A_SetRegister, QADC64_A_GetRegister	QSMCM register selection switch is invalid
QADC64_A_ERR_BAD_RESULT_ADDR	QADC64_A_GetRegister	Return address is NULL
QADC64_A_ERR_INVALID_DATA_VALUE	QADC64_A_SetRegister	Register value too large for register size
QADC64_A_ERR_INVALID_IARB	QADC64_A_Init	Invalid Interrupt Arbitration Number
QADC64_A_ERR_INVALID_TRIGGER_ASSIGNMENT	QADC64_A_Init	Invalid External Trigger Assignment
QADC64_A_ERR_INVALID_QCLK_PSH	QADC64_A_Init	Invalid QCLK Prescaler Clock High Time
QADC64_A_ERR_INVALID_QCLK_PSL	QADC64_A_Init	Invalid QCLK Prescaler Clock Low Time
QADC64_A_ERR_INVALID_IRL	QADC64_A_QueueInit	Invalid Interrupt Request Level
QADC64_A_ERR_INVALID_BQ2	QADC64_A_QueueInit	Invalid Beginning of Queue 2
QADC64_A_ERR_INVALID_CCW	QADC64_A_SetCCW	Invalid Conversion Command Word
QADC64_A_ERR_INVALID_IST	QADC64_A_SetCCW	Invalid Input Sample Time
QADC64_A_ERR_INVALID_CHANNEL	QADC64_A_SetCCW	Invalid Channel Number
QADC64_A_ERR_INVALID_MODE	QADC64_A_ControlEnable	Specified Queue Operating Mode is invalid
QADC64_A_ERR_INVALID_RESET	QADC64_A_ResetStatus	Specified Status Reset Flags are invalid
QADC64_A_ERR_INVALID_RESULT_WORD	QADC64_A_GetResultWord	Table entry is invalid
QADC64_A_ERR_INVALID_DATA_FORMAT	QADC64_A_GetResultWord	Data Format is invalid
QADC64_A_ERR_INVALID_PIN	QADC64_A_SetPin, QADC64_A_GetPin	Port Pin is invalid

14.4 QADC64_A API Definitions

14.4.1 QADC64_A_SetRegister

14.4.1.1 Description

QADC64_A_SetRegister writes a user-supplied data value to a user-specified QADC Register.

14.4.1.2 Prototype

```
QADC64_A_ReturnCode_t QADC64_A_SetRegister  (
          pQADC64_A_t                   QADCPtr,
          QADC64_A_RegisterSwitch_t     RegisterSwitch,
          UINT16                        RegisterValue
)
```

14.4.1.3 Arguments

Data Type	Formal Name	Description	Valid Values
pQADC64_A_t	QADCPtr	QADC64 base address associated with this driver.	Any non-zero core processor data address.
QADC64_A_RegisterSwitch_t	RegisterSwitch	Select among QADC registers.	QADC64_A_QADCMCR_SWITCH, QADC64_A_QADCINT_SWITCH, QADC64_A_PORTQA_SWITCH, QADC64_A_PORTQB_SWITCH, QADC64_A_DDRQA_SWITCH, QADC64_A_QACR0_SWITCH, QADC64_A_QACR1_SWITCH, QADC64_A_QACR2_SWITCH, QADC64_A_QASR0_SWITCH
UINT16	RegisterValue	Data to copy into selected QADC register.	Right-justified zero-extended 8-bit value for PORT A or PORTB. Any 16-bit value for all other registers.

14.4.1.4 Return Values

Error Code	Description
QADC64_A_ERR_NONE	No error
QADC64_A_ERR_INVALID_HANDLE	Base address parameter is zero
QADC64_A_ERR_INVALID_REGISTER	Register selection switch is invalid
QADC64_A_ERR_INVALID_DATA_VALUE	Register value too large for register size

14.4.2 QADC64_A_GetRegister

14.4.2.1 Description

Gets a QADC register. Parameter checking is a compile-time option.

14.4.2.2 Prototype

```
QADC64_A_ReturnCode_t QADC64_A_GetRegister (
       pQADC64_A_t                QADCPtr,
       QADC64_A_RegisterSwitch_t  RegisterSwitch,
       UINT16                    *GetRegisterPtr
)
```

14.4.2.3 Arguments

Data Type	Formal Name	Description	Valid Values
pQADC64_A_t	QADCPtr	QADC base address associated with this driver.	Any non-zero core processor data address.
QADC64_A_RegisterSwitch_t	RegisterSwitch	Select among QADC registers.	QADC64_A_QADCMCR_SWITCH, QADC64_A_QADCINT_SWITCH, QADC64_A_PORTQA_SWITCH, QADC64_A_PORTQB_SWITCH, QADC64_A_DDRQA_SWITCH, QADC64_A_QACR0_SWITCH, QADC64_A_QACR1_SWITCH, QADC64_A_QACR2_SWITCH, QADC64_A_QASR0_SWITCH, QADC64_A_QASR1_SWITCH
UINT16 *	GetRegisterPtr	Result address for selected QADC register.	Any non-zero core processor data address.

14.4.2.4 Return Values

Error Code	Description
QADC64_A_ERR_NONE	No error
QADC64_A_ERR_INVALID_HANDLE	Base address parameter is zero
QADC64_A_ERR_BAD_RESULT_ADDR	Result pointer is zero
QADC64_A_ERR_INVALID_REGISTER	Register selection switch is invalid

14.4.3 QADC64_A_Init

14.4.3.1 Description

QADC64_A_Init initializes the QADC. Parameter checking is a compile-time option.

14.4.3.2 Prototype

```
QADC64_A_ReturnCode_t QADC64_A_Init (
            pQADC64_A_t                   QADCPtr,
            QADC64_A_Boolean_t            StopEnable,
            QADC64_A_Boolean_t            FreezeEnable,
            QADC64_A_Boolean_t            SupervisorOnly,
            UINT8                         IARB,
            UINT8                         PortDataDirection,
            QADC64_A_Boolean_t            ExternalMuxMode,
            QADC64_A_TriggerAssignment_t  TriggerAssignment,
            UINT8                         PrescalerHighTime,
            UINT8                         PrescalerLowTime
)
```

14.4.3.3 Arguments

Data Type	Formal Name	Description	Valid Values
pQADC64_A_t	QADCPtr	Base address associated with this driver.	Any non-zero core processor data address.
QADC64_A_Boolean_t	StopEnable	Determines module behavior in Stop mode.	TRUE = enable Stop mode FALSE = disable Stop mode
QADC64_A_Boolean_t	FreezeEnable	Determines action of QADC when IMB FREEZE signal is asserted.	TRUE = freeze FALSE = ignore Freeze signal
QADC64_A_Boolean_t	SupervisorOnly	Determines whether module registers are accessible in supervisor-only mode or unrestricted	TRUE = supervisor-only FALSE = unrestricted
UINT8	IARB	Interrupt Arbitration Number	1..15
UINT8	PortDataDirection	Configures Port A pins as inputs or outputs	Any 8-bit value (1 = output, 0 = input).
QADC64_A_Boolean_t	ExternalMuxMode	Determines if QADC operates in the externally multiplexed mode.	TRUE = externally multiplexed FALSE = internally multiplexed
QADC64_A_TriggerAssignment_t	TriggerAssignment	Determines ETRIG[1] and ETRIG[2] pin trigger assignments.	ETRIG1_QUEUE1 = ETRIG[1] triggers queue1, ETRIG[2] triggers queue2 ETRIG1_QUEUE2 = ETRIG[1] triggers queue2, ETRIG[2] triggers queue1
UINT8	PrescalerHighTime	Selects QCLK high time	0..31
UINT8	PrescalerLowTime	Selects QCLK low time	0..7

14.4.3.4 Return Values

Error Code	Description
QADC64_A_ERR_NONE	No error
QADC64_A_ERR_INVALID_HANDLE	Base address parameter is zero
QADC64_A_ERR_INVALID_IARB	Invalid Interrupt Arbitration Number
QADC64_A_ERR_INVALID_QCLK_PSH	Invalid QCLK Prescaler Clock High Time
QADC64_A_ERR_INVALID_QCLK_PSL	Invalid QCLK Prescaler Clock Low Time
QADC64_A_ERR_INVALID_TRIGGER_ASSIGNMENT	Invalid External Trigger Assignment

14.4.4 QADC64_A_QueueInit

14.4.4.1 Description

QADC64_A_QueueInit initializes the operation of one QADC Queue. Parameter checking is a compile-time option.

14.4.4.2 Prototype

```
QADC64_A_ReturnCode_t QADC64_A_QueueInit (
        pQADC64_A_t           QADCPtr,
        QADC64_A_Boolean_t    Queue2Select,
        QADC64_A_Boolean_t    CompletionInterruptEnable,
        QADC64_A_Boolean_t    PauseInterruptEnable,
        UINT8                 InterruptRequestLevel,
        QADC64_A_Boolean_t    Queue2Resume,
        UINT8                 BeginningofQueue2
)
```

14.4.4.3 Arguments

Data Type	Formal Name	Description	Valid Values
pQADC64_A_t	QADCPtr	Base address associated with this driver.	Any non-zero core processor data address.
QADC64_A_Boolean_t	Queue2Select	Specifies queue to be initialized.	TRUE = queue 2 FALSE = queue 1
QADC64_A_Boolean_t	CompletionInterruptEnable	Enables an interrupt upon completion of the last CCW in queue.	TRUE = enable interrupt FALSE = disable interrupt
QADC64_A_Boolean_t	PauseInterruptEnable	Enables an interrupt upon completion of a CCW that has the pause bit set.	TRUE = enable interrupt FALSE = disable interrupt
UINT8	InterruptRequestLevel	Queue interrupt request level	0..31
QADC64_A_Boolean_t	Queue2Resume	Determines queue 2 behavior after suspension due to queue 1.	TRUE = resume execution with aborted CCW in queue 2 FALSE = begin execution with the first CCW in queue 2 or the current subqueue.
UINT8	BeginningofQueue2	Determines location where queue 2 begins.	0..127

14.4.4.4 Return Values

Error Code	Description
QADC64_A_ERR_NONE	No error
QADC64_A_ERR_INVALID_HANDLE	Base address parameter is zero
QADC64_A_ERR_INVALID_IRL	Invalid Interrupt Request Level
QADC64_A_ERR_INVALID_BQ2	Invalid Beginning of Queue 2

14.4.5 QADC64_A_SetCCW

14.4.5.1 Description

QADC64_A_SetCCW sets a QADC Conversion Command Word (CCW). Parameter checking is a compile-time option.

14.4.5.2 Prototype

```
QADC64_A_ReturnCode_t QADC64_A_SetCCW   (
        pQADC64_A_t           QADCPtr,
        UINT8                 CCW,
        QADC64_A_Boolean_t    Pause,
        UINT8                 InputSampleTime,
        UINT8                 ChannelNumber
)
```

14.4.5.3 Arguments

Data Type	Formal Name	Description	Valid Values
pQADC64_A_t	QADCPtr	Base address associated with this driver.	Any non-zero core processor data address.
UINT8	CCW	Specifies an entry in the CCW table.	0..63
QADC64_A_Boolean_t	Pause	Specifies the Pause bit for the CCW.	TRUE = enter the pause state after execution of the current CCW. FALSE = do not enter the pause state after execution of the current CCW.
UINT8	InputSampleTime	Specifies the length of the Input Sample Time window.	0..3
UINT8	ChannelNumber	Specifies the input channel number.	0..31 or 48..63

14.4.5.4 Return Values

Error Code	Description
QADC64_A_ERR_NONE	No error
QADC64_A_ERR_INVALID_HANDLE	Base address parameter is zero
QADC64_A_ERR_INVALID_CCW	Invalid CCW
QADC64_A_ERR_INVALID_IST	Invalid Input Sample Time
QADC64_A_ERR_INVALID_CHANNEL	Invalid Channel Number

14.4.6 QADC64_A_GetStatus

14.4.6.1 Description

QADC64_A_GetStatus returns the QADC status. Parameter checking is a compile-time option.

14.4.6.2 Prototype

```
QADC64_A_ReturnCode_t QADC64_A_GetStatus  (
        pQADC64_A_t          QADCPtr,
        QADC64_A_Boolean_t   Queue2Select,
        pQADC64_A_Status_t   StatusPtr
)
```

14.4.6.3 Arguments

Data Type	Formal Name	Description	Valid Values
pQADC64_A_t	QADCPtr	QADC base address associated with this driver.	Any non-zero core processor data address.
QADC64_A_Boolean_t	Queue2Select	Specifies queue status to be returned.	TRUE = queue 2 FALSE = queue 1
pQADC64_A_Status_t	StatusPtr	Result address for specified queue completion status	Any non-zero core processor data address.

14.4.6.4 Return Values

Error Code	Description
QADC64_A_ERR_NONE	No error
QADC64_A_ERR_INVALID_HANDLE	Base address parameter is zero
QADC64_A_ERR_BAD_RESULT_ADDR	Result pointer is zero

14.4.7 QADC64_A_ControlEnable

14.4.7.1 Description

QADC64_A_ControlEnable enables or disables QADC conversion. Parameter checking is a compile-time option.

14.4.7.2 Prototype

```
QADC64_A_ReturnCode_t QADC64_A_ControlEnable (
          pQADC64_A_t           QADCPtr,
          QADC64_A_Boolean_t    Queue2Select,
          QADC64_A_Boolean_t    SingleScanEnable,
          UINT8                 OperatingMode
)
```

14.4.7.3 Arguments

Data Type	Formal Name	Description	Valid Values
pQADC64_A_t	QADCPtr	QADC base address associated with this driver.	Any non-zero core processor data address.
QADC64_A_Boolean_t	Queue2Select	Specifies queue status to be controlled.	TRUE = queue 2 FALSE = queue 1
QADC64_A_Boolean_t	SingleScanEnable	Enable a single-scan of specified queue.	TRUE = Accept single-scan mode trigger events. FALSE = Single-scan mode trigger events are not accepted.
UINT8	OperatingMode	Operating Mode for specified queue.	0..31

14.4.7.4 Return Values

Error Code	Description
QADC64_A_ERR_NONE	No error
QADC64_A_ERR_INVALID_HANDLE	Base address parameter is zero
QADC64_A_ERR_INVALID_MODE	Specified Queue Operating Mode is invalid

14.4.8 QADC64_A_ResetStatus

14.4.8.1 Description

QADC64_A_ControlEnable resets the specified QADC status flags. Parameter checking is a compile-time option.

14.4.8.2 Prototype

```
QADC64_A_ReturnCode_t QADC64_A_ResetStatus (
          pQADC64_A_t    QADCPtr,
          UINT16         ResetFlags
)
```

14.4.8.3 Arguments

Data Type	Formal Name	Description	Valid Values
pQADC64_A_t	QADCPtr	QADC base address associated with this driver.	Any non-zero core processor data address.
UINT16	ResetFlags	Specifies queue status to be reset.	Logical OR of any of: QASR0_CF1_MASK, QASR0_PF1_MASK, QASR0_TOR1_MASK, QASR0_CF2_MASK, QASR0_PF2_MASK, QASR0_TOR2_MASK

14.4.8.4 Return Values

Error Code	Description
QADC64_A_ERR_NONE	No error
QADC64_A_ERR_INVALID_HANDLE	Base address parameter is zero
QADC64_A_ERR_INVALID_RESET	Specified Status Reset Flags are invalid

14.4.9 QADC64_A_GetResultWord

14.4.9.1 Description

QADC64_A_GetResultWord reads an entry in the QADC Result Word Table. Parameter checking is a compile-time option.

14.4.9.2 Prototype

```
QADC64_A_ReturnCode_t QADC64_A_GetResultWord (
          pQADC64_A_t              QADCPtr,
          UINT8                    ResultWord,
          QADC64_A_DataFormat_t    DataFormat,
          UINT16                  *GetResultWordPtr
)
```

14.4.9.3 Arguments

Data Type	Formal Name	Description	Valid Values

pQADC64_A_t	QADCPtr	QADC base address associated with this driver.	Any non-zero core processor data address.
UINT8	ResultWord	Specifies an entry in the Result Word Table.	0..63
QADC64_A_DataFormat_t	DataFormat	Specifies the Result Word data format	QADC64_A_RJURR_DATA_FORMAT, QADC64_A_LJSRR_DATA_FORMAT, QADC64_A_LJURR_DATA_FORMAT
UINT16 *	GetResultWordPtr	Result address for selected QADC register.	Any non-zero core processor data address.

14.4.9.4 Return Values

Error Code	Description
QADC64_A_ERR_NONE	No error
QADC64_A_ERR_INVALID_HANDLE	Base address parameter is zero
QADC64_A_ERR_BAD_RESULT_ADDR	Result pointer is zero
QADC64_A_ERR_INVALID_RESULT_WORD	Result Word Table entry is invalid
QADC64_A_ERR_INVALID_DATA_FORMAT	Result Word Data Format is invalid

14.4.10 QADC64_A_SetPin

14.4.10.1 Description

QADC64_A_SetPin sets a QADC Port A Pin. QADC Port B Pins are input-only. Parameter checking is a compile-time option.

14.4.10.2 Prototype

```
QADC64_A_ReturnCode_t QADC64_A_SetPin (
          pQADC64_A_t           QADCPtr,
          UINT16                PortPinSwitch,
          QADC64_A_Boolean_t    PinLogic
)
```

14.4.10.3 Arguments

Data Type	Formal Name	Description	Valid Values
pQADC64_A_t	QADCPtr	QADC64 base address associated with this driver.	Any non-zero core processor data address.
UINT16	PortPinSwitch	Select among QADC Port A Pins.	PORTQA_PQA7_MASK, PORTQA_PQA6_MASK, PORTQA_PQA5_MASK, PORTQA_PQA4_MASK, PORTQA_PQA3_MASK, PORTQA_PQA2_MASK, PORTQA_PQA1_MASK, PORTQA_PQA0_MASK
QADC64_A_Boolean_t	PinLogic	Logic to be driven onto selected Port A Pin.	TRUE = Logic 1 FALSE = Logic 0

14.4.10.4 Return Values

Error Code	Description
QADC64_A_ERR_NONE	No error
QADC64_A_ERR_INVALID_HANDLE	Base address parameter is zero
QADC64_A_ERR_INVALID_PIN	Port Pin is invalid

14.4.11 QADC64_A_GetPin

14.4.11.1 Description

QADC64_A_GetPin gets a QADC Port Pin. Parameter checking is a compile-time option.

14.4.11.2 Prototype

```
QADC64_A_ReturnCode_t QADC64_A_GetPin (
       pQADC64_A_t           QADCPtr,
       UINT16                PortPinSwitch,
       QADC64_A_Boolean_t   *PortPinPtr
)
```

14.4.11.3 Arguments

Data Type	Formal Name	Description	Valid Values
pQADC64_A_t	QADCPtr	QADC base address associated with this driver.	Any non-zero core processor data address.
UINT16	PortPinSwitch	Select among QADC Port A/B Pins.	PORTQA_PQA7_MASK, PORTQA_PQA6_MASK, PORTQA_PQA5_MASK, PORTQA_PQA4_MASK, PORTQA_PQA3_MASK, PORTQA_PQA2_MASK, PORTQA_PQA1_MASK, PORTQA_PQA0_MASK, PORTQB_PQB7_MASK, PORTQB_PQB6_MASK, PORTQB_PQB5_MASK, PORTQB_PQB4_MASK, PORTQB_PQB3_MASK, PORTQB_PQB2_MASK, PORTQB_PQB1_MASK, PORTQB_PQB0_MASK
QADC64_A_Boolean_t *	PortPinPtr	Result address for selected QADC PORT A/B Pin.	Any non-zero core processor data address.

14.4.11.4 Return Values

Error Code	Description
QADC64_A_ERR_NONE	No error
QADC64_A_ERR_INVALID_HANDLE	Base address parameter is zero
QADC64_A_ERR_BAD_RESULT_ADDR	Result pointer is zero
QADC64_A_ERR_INVALID_PIN	Port Pin is invalid

14.5 QADC64_A Example Application

The following describes the QADC64_A example application.

14.5.1 Description for Example Application

The example application illustrates the use of the QADC64_A Level 1 driver. This example program will have the QADC64 module do 10 conversions using Queue 1, and then calculate the average result. If an error occurs, the program will return with the line number of where the failure occurred.

14.5.1.1 Example Application Code

```
/*******************************************************************

File:    qadc64a_app.c

Purpose: The following example code illustrates the use of the M-CORE
         Peripherals Library QADC64_A Level 1 driver.

         This program illustrates use of the QADC64_A Level 1 driver.
         This example program will have the QADC64 module do 10
         conversions using Queue 1, and then calculate the average
         result.

         If an error occurs, the program will return with the line
         number of where the failure occurred.

Time to run:
         Instantaneous.

General Setup:
  General hardware and software setup to run this application
  can be found in the 'readme_app.txt' file, located in the
  '<library>/app' directory.

Application Specific Setup:
         Development Board: MMC2107.
         The program requires the QADC64 pin VRL be tied to Ground,
         the QADC64 pin VRH be tied to 5 volts and the QADC64 pin
         1PQB0 be tied to a variable voltage source.

(C) Copyright Motorola Inc, 2000.  All rights reserved.
*******************************************************************/

#include "qadc64_a.h"
#include <stdio.h>
#include "plibdefs_mmc2107.h"

#define QADC64_CONVERSION_QUANTITY 10 /* Number of QADC64 Conversions*/

/* macro to handle error checking and response */
```

```
#define handle_error(err)   if (err != QADC64_A_ERR_NONE) return(__LINE__);

/* global definitions */
pQADC64_A_t qadc64 = (pQADC64_A_t) __MMC2107_QADC64;
QADC64_A_ReturnCode_t retval = QADC64_A_ERR_NONE;

/* Function prototypes                                                */
int main(void);
int qadc64_example(pQADC64_A_t);

/*********************************************************************
   Name: main
   Description: Drive program operation
   Inputs: none
   Outputs: none
*********************************************************************/
int main ()
{
   volatile int status;

   status = qadc64_example ((pQADC64_A_t)__MMC2107_QADC64);

   return (status);

}   /* end main */

/*********************************************************************
   Name: qadc64_example
   Description: Display QADC64 operation
   Inputs: qadc64 (pointer to QADC64_A base address)
   Outputs: return code status
*********************************************************************/
int qadc64_example (pQADC64_A_t qadc64)
{
    QADC64_A_Status_t status;    /* QADC64 status structure          */
    UINT16 result;                /* QADC64 result word address       */
    volatile UINT16 averageresult;   /* QADC64 result word average       */
    UINT8 i;                     /* counter                        */

    /* Configure the QADC64 module                                   */
    retval = QADC64_A_Init(qadc64,  /* QADC device handle            */
                           QADC64_A_FALSE,   /* Ignore the Stop Signal        */
                           QADC64_A_FALSE,   /* Ignore the Freeze Signal      */
                           QADC64_A_FALSE,   /* Unrestricted module access    */
                           4,       /* Use IARB = 4                  */
                           0,       /* Configure Port A pins as input*/
                           QADC64_A_FALSE,   /* Internally multiplexed mode   */
                           ETRIG1_QUEUE1, /* ETRIG1 to Queue 1       */
                           3,       /* Prescaler High Time           */
                           3        /* Prescaler Low Time            */
                          );
    handle_error(retval);

    /* Initialize QADC64 Queue 1                                     */
```

QADC64_A Level 1

```
    retval = QADC64_A_QueueInit(qadc64, /* QADC64 device handle      */
                                QADC64_A_FALSE,    /* Use Queue 1 in this prog. */
                                QADC64_A_FALSE,    /* Disable Completion ints   */
                                QADC64_A_FALSE,    /* Disable Pause interrupts  */
                                QADC64_A_FALSE,    /* Queue 2 is unused         */
                                63        /* Queue 2 is unused         */
                               );
    handle_error(retval);

    /* set CCWs 0 to 8 with pause bit cleared                          */
    for (i=0; i < (QADC64_CONVERSION_QUANTITY -1); i++)
    {
        /* set CCW */
        retval = QADC64_A_SetCCW(qadc64, /* QADC64 device handle     */
                                 i,        /* CCW                     */
                                 QADC64_A_FALSE,    /* don't set the Pause    */
                                 0,        /* Input Sample Time       */
                                 0         /* Channel 0 = pin 1PQB0   */
                                );
        handle_error(retval);

    }

    /* set 10th CCW with pause bit set                                 */
    retval = QADC64_A_SetCCW(qadc64,      /* QADC64 device handle    */
                             (QADC64_CONVERSION_QUANTITY-1), /* CCW  */
                             QADC64_A_TRUE,        /* pause on the final CCW  */
                             0,           /* Input Sample Time       */
                             0            /* Channel 0 = pin 1PQB0   */
                            );
    handle_error(retval);

    /* enable the QADC64 module                                        */
    retval = QADC64_A_ControlEnable(qadc64,   /* QADC64 device handle*/
                                    QADC64_A_FALSE,    /* Use Queue 1         */
                                    QADC64_A_TRUE,     /* Do a Single-Scan    */
                                    1         /* Software Triggered  */
                                              /* Single-Scan mode    */
                                   );
    handle_error(retval);

    /* Poll until Queue 1 Completion Flag transitions to set     */
    do
    {
        retval = QADC64_A_GetStatus(qadc64, /* QADC64 device handle  */
                                    QADC64_A_FALSE,  /* Request Queue 1 Status*/
                                    &status /* StatusPtr             */
                                   );
        handle_error(retval);

    } while ( status.CompletionFlag != QADC64_A_TRUE );/* end while loop      */

    /* Reset QADC64 Queue 1 Completion Flag status bit                 */
    retval = QADC64_A_ResetStatus(qadc64,    /* QADC64 device handle */
```

```
                                QASR0_CF1_MASK/* Reset Queue 1 CF1 */
                               );
    handle_error(retval);

    /* Get QADC64 Result Words                                          */
    averageresult = 0;        /* Clear Result Word average  variable */
    for (i=0; i < QADC64_CONVERSION_QUANTITY; i++)
    {
        retval = QADC64_A_GetResultWord(qadc64,
                                        /* QADC64 device handle */
                                    i,      /* Result Word entry */
                                    QADC64_A_RJURR_DATA_FORMAT,
                                        /* Format            */
                                    &result /* result word       */
                                   );
        handle_error(retval);

        averageresult += result;     /* add result to average result */
    }

    /* Step over this line to see the value in averageresult. This   */
    /* value should equal the voltage present on the QADC64 pin 1PQB0*/

    /* calculate integer average                                          */
    averageresult = (UINT16)(averageresult / QADC64_CONVERSION_QUANTITY);

    return (retval);

}   /* end QADC64 Example Program */
```

QADC64_A Level 1

Section 15 Reset_A Level 1

This section describes the Reset_A Level 1 M•CORE™ Device Driver.

15.1 Overview

The Reset Controller and its accompanying driver (Reset_A) are used on the MMC2107 development system. The level 1 service is designed to allow the programmer to control the Reset Controller on a near-register level, but without the concern of exact positions of registers and fields, or the detailed sequences of register operations. Refer to the MMC2107 Advance Information (Data Book) (MMC2107/D) for additional information.

15.2 Reset_A Service Functions

The Reset_A device driver software provides these functions:

- Allows independent or software requested asserting of the external reset pin.
- Allows software to request a reset.
- Returns the reset status of the modules.
- Writes unformatted data to the Reset Control Register (RCR).
- Gets unformatted data from a specified reset module register.

15.3 Reset_A Data Type Definitions

The following paragraphs describe the data definitions recognized by the Reset_A device driver. These data definitions are grouped into six categories: standard data types, board specific base address definitions, derived data types, enumerated data types, register macros, and return codes.

15.3.1 Standard Data Type Definitions

- UINT8 — 8-bit unsigned data value
- INT8 — 8-bit signed data value
- UINT16 — 16-bit unsigned data value
- INT16 — 16-bit signed data value
- UINT32 — 32-bit unsigned data value
- INT32 — 32-bit signed data value

15.3.2 Board Specific Base Address Definitions

```
#define __MMC2107_RESET   (0x00C4 0000)
```

15.3.3 Derived Data Type Definitions

15.3.3.1 RESET_A_StatusRegister_t - Reset Status Register Definition

```
typedef struct {
          Reset_A_Boolean_t  SoftwareResetFlag;           /* SR-EOSF */
          Reset_A_Boolean_t  WatchdogTimerResetFlag;      /* SR-RLDF */
          Reset_A_Boolean_t  PowerOnResetFlag;            /* SR-SFO  */
          Reset_A_Boolean_t  ExternalResetFlag;           /* SR-NFO  */
          Reset_A_Boolean_t  LossOfClockResetFlag;        /* SR-FIFO */
          Reset_A_Boolean_t  LossOfLockResetFlag;         /* SR-FIFO */
} RESET_A_StatusRegister_t, *pRESET_A_StatusRegister_t;
```

15.3.4 Enumerated Data Type Definitions

15.3.4.1 Reset_A_Boolean_t - Reset Module Boolean Enum

```
typedef enum{
  Reset_A_FALSE,          /* Logical value FALSE  */
  Reset_A_TRUE            /* Logical value TRUE   */
} Reset_A_Boolean_t;      /* Reset_A boolean enum */
```

15.3.4.2 Reset_A_Register_t - Reset Module Register Selection

```
typedef enum {
  Reset_A_RCR,         /* Select RESET Control Register     */
  Reset_A_RSR          /* Select RESET Status Register      */
} Reset_A_Register_t;
```

15.3.4.3 Reset_A_ReturnCode_t - Reset Module Return Codes

```
typedef enum {
  RESET_A_ERR_NONE,
  RESET_A_ERR_INVALID_HANDLE,
  RESET_A_ERR_BAD_RESULT_ADDR,
  RESET_A_ERR_INVALID_REGISTER,
  RESET_A_ERR_INVALID_EXTERNAL_RESET_PARAMETER,
  RESET_A_ERR_INVALID_SOFT_RESET_PARAMETER
} Reset_A_ReturnCode_t;
```

15.3.4.4 Reset_A_SoftwareResetRequest_t - Soft Reset Selection

```
typedef enum {
  Reset_A_NO_SOFTWARE_REQUEST,      /* Select RESET Control Register   */
  Reset_A_REQUEST_SOFTWARE_RESET    /* Select RESET Status Register    */
} Reset_A_SoftwareResetRequest_t;
```

15.3.4.5 Reset_A_Control_t - Reset Module Register Selection

```
typedef enum {
  Reset_A_ENABLE,     /* Select RESET Control Register  */
  Reset_A_DISABLE     /* Select RESET Status Register   */
} Reset_A_Control_t;
```

15.3.5 Register Macros

15.3.5.1 Control Register

```
#define RCR_SOFTRST_BITNO 7
#define RCR_SOFTRST_MASK (1 << RCR_SOFTRST_BITNO)
#define RCR_FRCRSTOUT_BITNO 6
#define RCR_FRCRSTOUT_MASK (1 << RCR_FRCRSTOUT_BITNO)
```

15.3.5.2 Status Register

```
#define RSR_SOFT_BITNO 5
#define RSR_SOFT_MASK (1 << RSR_SOFT_BITNO)
#define RSR_WDR_BITNO 4
#define RSR_WDR_MASK (1 << RSR_WDR_BITNO)
#define RSR_POR_BITNO 3
#define RSR_POR_MASK (1 << RSR_POR_BITNO)
#define RSR_EXT_BITNO 2
#define RSR_EXT_MASK (1 << RSR_EXT_BITNO)
#define RSR_LOC_BITNO 1
#define RSR_LOC_MASK (1 << RSR_LOC_BITNO)
#define RSR_LOL_BITNO 0
#define RSR_LOL_MASK (1 << RSR_LOL_BITNO)
```

15.3.6 Return Codes

Return codes from the enumerated type “Reset_A_ReturnCode_t”.

Return Code	Functions Returning	Description
RESET_A_ERR_NONE	All	No error.
RESET_A_ERR_INVALID_HANDLE	All	RESET base address parameter is zero.
RESET_A_ERR_BAD_RESULT_ADDR	GetRegister, GetResetStatus	Result Pointer is zero.
RESET_A_ERR_INVALID_REGISTER	SetRegister, GetRegister	RESET Register Selection switch is invalid.
RESET_A_ERR_INVALID_EXTERNAL_RESET_PARAMETER	Reset_A_ControlExternalReset	The external reset value is out of range.
RESET_A_ERR_INVALID_SOFT_RESET_PARAMETER	Reset_A_SoftReset	The soft reset value is out of range.

15.4 Reset_A API Definitions

15.4.1 Reset_A_ControlExternalReset

15.4.1.1 Description

The RESET_A_ControlExternalReset functions allows software control for requesting a reset or for independently asserting the external reset pin.

This function can be called at any time.

15.4.1.2 Prototype

```
Reset_A_ReturnCode_t  Reset_A_ControlExternalReset(
          pReset_A_t           RESETPtr,
          Reset_A_Control_t    ExternalResetControlVal
)
```

15.4.1.3 Arguments

Data Type	Formal Name	Description	Valid Values
pReset_A_t	RESETPtr	Reset module base address associated with this driver.	Any non-zero core processor data address.
Reset_A_Control_t	ExternalResetControlVal	External reset control bit	Reset_A_ENABLE Reset_A_DISABLE

15.4.1.4 Return Values

Error Code	Description
RESET_A_ERR_NONE	No error
RESET_A_ERR_INVALID_HANDLE	Reset module base address parameter is zero.
RESET_A_ERR_INVALID_EXTERNAL_RESET_PARAMETER	The external reset value is out of range.

15.4.2 Reset_A_SoftReset

15.4.2.1 Description

The Reset_A_SoftReset function allows software to request a reset.

This function can be called at any time.

15.4.2.2 Prototype

```
Reset_A_ReturnCode_t  Reset_A_SoftReset(
          pReset_A_t    RESETPtr
)
```

15.4.2.3 Arguments

Data Type	Formal Name	Description	Valid Values
pReset_A_t	RESETPtr	Reset module base address associated with this driver.	Any non-zero core processor data address.

15.4.2.4 Return Values

Error Code	Description
RESET_A_ERR_NONE	No error
RESET_A_ERR_INVALID_HANDLE	Reset module base address parameter is zero.

15.4.3 Reset_A_GetResetStatus

15.4.3.1 Description

The Reset_A_GetResetStatus function returns the reset status from the Reset Status Register (RSR).

This function can be called at any time.

15.4.3.2 Prototype

```
Reset_A_ReturnCode_t  Reset_A_GetResetStatus (
          pReset_A_t                  RESETPtr,
          RESET_A_StatusRegister_t   *StatusPtr
)
```

15.4.3.3 Arguments

Data Type	Formal Name	Description	Valid Values
pReset_A_t	RESETPtr	RESET base address associated with this driver.	Any non-zero core processor data address.
RESET_A_StatusRegister_t	StatusPtr	Result address for specified queue completion status	Any non-zero core processor data address.

15.4.3.4 Return Values

Error Code	Description
RESET_A_ERR_NONE	No error
RESET_A_ERR_INVALID_HANDLE	RESET base address parameter is zero
RESET_A_ERR_BAD_RESULT_ADDR	Result pointer is zero

15.4.4 Reset_A_SetRegister

15.4.4.1 Description

The Reset_A_SetRegister function writes unformatted data to the Reset Control Register (RCR).

This function can be called at any time.

15.4.4.2 Prototype

```
Reset_A_ReturnCode_t  Reset_A_SetRegister (
          pReset_A_t    RESETPtr,
          UINT8         RegisterValue
)
```

15.4.4.3 Arguments

Data Type	Formal Name	Description	Valid Values
pReset_A_t	RESETPtr	Reset_A base address associated with this driver.	Any non-zero core processor data address.
UINT8	RegisterValue	Data to be written to the RCR register.	Any 8-bit value.

15.4.4.4 Return Values

Error Code	Description
RESET_A_ERR_NONE	No error
RESET_A_ERR_INVALID_HANDLE	RESET base address parameter is zero

15.4.5 Reset_A_GetRegister

15.4.5.1 Description

The Reset_A_GetRegister function returns unformatted data from a specified Reset Controller register.

This function can be called at any time.

15.4.5.2 Prototype

```
Reset_A_ReturnCode_t  Reset_A_GetRegister (
          pReset_A_t             RESETPtr,
          Reset_A_RegSwitch_t    RESETRegisterSwitch,
          UINT8                 *GetRegisterPtr
)
```

15.4.5.3 Arguments

Data Type	Formal Name	Description	Valid Values
pReset_A_t	RESETPtr	Reset_A base address associated with this driver.	Any non-zero core processor data address.
Reset_A_Register_t	RESETRegSwitch	Select among a sub-set of Reset_A registers.	Reset_A_RCR Reset_A_RSR
UINT8 *	GetRegisterPtr	Result address for selected Reset_A register data.	Any non-zero core processor data address.

15.4.5.4 Return Values

Error Code	Description
RESET_A_ERR_NONE	No error
RESET_A_ERR_INVALID_HANDLE	RESET base address parameter is zero
RESET_A_ERR_BAD_RESULT_ADDR	Result Pointer is zero
RESET_A_ERR_INVALID_REGISTER	RESET Register Selection switch is invalid

15.5 Reset_A Example Application

The following describes the Reset_A example application.

15.5.1 Description for Example Application

The example program obtains the status of every reset source.

15.5.1.1 Example Application Code

```
/*********************************************************************

File:    reseta_app.c

Purpose: The following example code illustrates the use of the M-CORE
         Peripherals Library RESET_A Level 1 driver.

         The program obtains the status flags.

         If an error occurs, the program will exit with the line
         number of where the failure occurred.

Time to run:
         Less than 30 seconds.

General Setup:
         General hardware and software setup to run this application
         can be found in the 'readme_app.txt' file, located in the
         '<library>/app' directory.

Application Specific Setup:
         Development Board: MMC2107 CMB or EVB

(C) Copyright Metrowerks Inc, 2000.  All rights reserved.
*********************************************************************/

#include "reset_a.h"
#include "plibdefs_mmc2107.h"

int main(void);                /* function prototype */
int reset_example(pRESET_A_t);     /* function prototype */
#define RESET_A_PARAM_CHECKING   1

/*********************************************************************
   Name: main
   Description: Drive program operation
   Inputs: resetptr (pointer to RESET_A base address)
   Outputs: none
*********************************************************************/
int main ()
{
    int status;
```

```
    status = reset_example ((pRESET_A_t)__MMC2107_RESET);

      return (status);
}

/*********************************************************************
   Name: reset_example
   Description: Provide an example for the usage of the RESET_A
   Inputs:  resetptr    (pointer to RESET_A base address)
   Outputs: status (if passed, 0; else line number of the failure)
*********************************************************************/
int reset_example (pRESET_A_t resetptr)
{

/*Call Reset_A_GetResetStatus to obtain the status of the Reset module.  */
  RESET_A_StatusRegister_t  status;
  pRESET_A_StatusRegister_t status_p = &status;
  RESET_A_t  reset;


  resetptr = &reset;


  resetptr->RSR = 0xFF;  /* To Be removed when SIKA board is available */


  Reset_A_GetResetStatus (resetptr,            /* base address */
                          status_p       /* status structure */
                                  );

  return(0);

} /* end of main */
```

Reset_A Level 1

Section 16 SCI_D Level 1

This section describes the SCI_D Level 1 M•CORE™ Device Driver.

16.1 Overview

The Serial Communications Interface (SCI) and its accompanying driver (SCI_D) are used on the MMC2107 development system. The level 1 service is designed to allow the programmer to control the SCI on a near-register level, but without the concern of exact positions of registers and fields, or the detailed sequences of register operations. Refer to the MMC2107 Advance Information (Data Book) (MMC2107/D) for additional information.

16.2 SCI_D Service Functions

The SCI_D device driver software provides these functions:

- Initializes the SCI.
- Enables and disables the SCI transmitter and the SCI receiver.
- Enables and disables SCI interrupts.
- Writes data to the SCI data registers, SCIDRL and SCIDRH, and returns the transmitter status.
- Gets the contents of the SCI data registers, SCIDRL and SCIDRH, and gets the receiver status.
- Enables or disables the sending of break characters.
- Enables or disables the feedback path on the data serial shifter and sets the mode of loop operation based upon user input.
- Sets the baud rate of the SCI.
- Writes an 8-bit value to one of the SCI registers.
- Gets the contents of one of the SCI registers.
- Sets data direction for general purpose I/O (GPIO) pins, sets reduced drive status for GPIO pins, and enables, disables, or leaves unchanged pullups for GPIO pins.
- Gets the GPIO pin states.
- Changes GPIO pin states.
- Resets SCI registers to their reset (power on) values.
- Gets the status of the SCI.

16.3 SCI_D Data Type Definitions

The following paragraphs describe the data definitions recognized by the SCI_D device driver. These data definitions are grouped into seven categories: standard data types, board specific base address definitions, derived data types, enumerated data types, register macros, miscellaneous macros, and return codes.

16.3.1 Standard Data Type Definitions

- UINT8 — 8-bit unsigned data value
- INT8 — 8-bit signed data value
- UINT16 — 16-bit unsigned data value
- INT16 — 16-bit signed data value
- UINT32 — 32-bit unsigned data value
- INT32 — 32-bit signed data value

16.3.2 Board Specific Base Address Definitions

```
#define __MMC2107_SCI1   (0x00CC0000)
#define __MMC2107_SCI2   (0x00CD0000)
```

16.3.3 Derived Data Type Definitions

16.3.3.1 SCI_D_T – SCI Register Definition

```
typedef struct {
           UINT8   SCIBDH;          /* Baud Rate Register High          */
           UINT8   SCIBDL;          /* Baud Rate Register Low           */
           UINT8   SCICR1;          /* Control Register 1               */
           UINT8   SCICR2;          /* Control Register 2               */
  volatile UINT8   SCISR1;          /* Status Register 1                */
  volatile UINT8   SCISR2;          /* Status Register 2                */
  volatile UINT8   SCIDRH;          /* Data Register High               */
  volatile UINT8   SCIDRL;          /* Data Register Low                */
           UINT8   SCIPURD;         /* Pullup and Reduced Drive Register*/
  volatile UINT8   SCIPORT;         /* Port Data Register               */
           UINT8   SCIDDR;          /* Data Direction Register          */
} SCI_D_t, *pSCI_D_t;
```

16.3.3.2 SCI_D_Status_t – Status Structure Definition

```
typedef struct {
  /* Potential transmission errors */
         SCI_D_Boolean_t    transmitDataPending; /*data pending error when TRUE*/

 /* Potential reception errors */
         SCI_D_Boolean_t   receiveDataPending;  /*data pending error when TRUE*/
         SCI_D_Boolean_t   dataOverrun;   /*data overrun error occurred when TRUE*/
```

```
        SCI_D_Boolean_t   noiseDetected; /*input noise detected error when TRUE*/
        SCI_D_Boolean_t   framingError;  /*framing error occurred when TRUE    */
        SCI_D_Boolean_t   parityError;   /*parity error occurred when TRUE     */
        SCI_D_Boolean_t   receiverActive;/*receiver is busy when TRUE          */

 /* Other information          */
        SCI_D_Boolean_t    transmitInProgress; /*transmit in progress when TRUE*/
        SCI_D_Boolean_t    idleLine;           /*line is idle when TRUE*/
} SCI_D_Status_t, *pSCI_D_Status_t;
```

16.3.3.3 SCI_D_ReceiverStatus_t – Receiver Status Structure Definition

```
typedef struct {
 /* Potential reception errors */
       SCI_D_Boolean_t    receiveDataPending;  /*data pending error when TRUE    */
       SCI_D_Boolean_t    dataOverrun;    /*data overrun error occurred when TRUE*/
       SCI_D_Boolean_t    noiseDetected;  /*input noise detected error when TRUE */
       SCI_D_Boolean_t    framingError;   /*framing error occurred when TRUE     */
       SCI_D_Boolean_t    parityError;    /*parity error occurred when TRUE      */
       SCI_D_Boolean_t    receiverActive; /*receiver is busy when TRUE           */
} SCI_D_ReceiverStatus_t, *pSCI_D_ReceiverStatus_t;
```

16.3.3.4 SCI_D_DataDirStruct_t – Data Direction Structure Definition

```
typedef struct {
       SCI_D_DataDir_t    dataDirPin0;     /* port 0 data direction */
       SCI_D_DataDir_t    dataDirPin1;     /* port 1 data direction */
       SCI_D_DataDir_t    dataDirPin2;     /* port 2 data direction */
       SCI_D_DataDir_t    dataDirPin3;     /* port 3 data direction */
       SCI_D_DataDir_t    dataDirPin4;     /* port 4 data direction */
       SCI_D_DataDir_t    dataDirPin5;     /* port 5 data direction */
       SCI_D_DataDir_t    dataDirPin6;     /* port 6 data direction */
       SCI_D_DataDir_t    dataDirPin7;     /* port 7 data direction */
} SCI_D_DataDirStruct_t, *pSCI_D_DataDirStruct_t;
```

16.3.4 Enumerated Data Type Definitions

16.3.4.1 SCI_D_ReturnCode_t – SCI_D Error Codes

```
typedef enum
{
  SCI_D_ERR_NONE,                 /* no error                           */
  SCI_D_ERR_INVALID_HANDLE,       /* SCI base address parameter is zero */
  SCI_D_ERR_BAD_RESULT_ADDR,      /* Result pointer is zero             */
  SCI_D_ERR_INVALID_ADDR,         /* Pointer is zero                    */
  SCI_D_ERR_INVALID_REGISTER,     /* Register selection is invalid      */
  SCI_D_ERR_INVALID_CONTROL,      /* Control selection is invalid       */
  SCI_D_ERR_INVALID_BAUD_RATE,    /* Baud rate is invalid               */
  SCI_D_ERR_INVALID_CLOCK_VALUE,  /* Clock value is invalid             */
  SCI_D_ERR_INVALID_CLOCK_DIVIDER,/* Clock divider value is invalid     */
  SCI_D_ERR_INVALID_PIN_MODE,     /* Pin mode is invalid                */
  SCI_D_ERR_INVALID_IDLE_MODE,    /* Idle mode is invalid               */
```

```
  SCI_D_ERR_INVALID_PARITY,         /* Parity selection is invalid        */
  SCI_D_ERR_INVALID_DATA_FRAME,     /* Data frame selection is invalid    */
  SCI_D_ERR_INVALID_WAKEUP_MODE,    /* Wakeup mode selection is invalid   */
  SCI_D_ERR_INVALID_DATA_VALUE,     /* Data frame value is invalid        */
  SCI_D_ERR_INVALID_RECEIVER_SOURCE,/* Receiver source is invalid         */
  SCI_D_ERR_INVALID_DATA_DIRECTION, /* Data direction is invalid          */
  SCI_D_ERR_INVALID_DRIVE_MODE,     /* Drive mode is invalid              */
  SCI_D_ERR_INVALID_PORT_NUMBER,    /* Port number selection is invalid   */
  SCI_D_ERR_INVALID_PIN_SENSE,      /* Pin since is invalid               */
  SCI_D_ERR_WAKEUP_PARITY_CONFLICT  /* Address mark wakeup and parity are enabled*/
} SCI_D_ReturnCode_t;
```

16.3.4.2 SCI_D_Register_t – SCI_D Register Selection

```
typedef enum {

      SCI_D_SCIBDH,       /* Select Baud Rate Register High       */
      SCI_D_SCIBDL,       /* Select Baud Rate Register Low        */
      SCI_D_SCICR1,       /* Select Control Register 1            */
      SCI_D_SCICR2,       /* Select Control Register 2            */
      SCI_D_SCISR1,       /* Select Status Register 1             */
      SCI_D_SCISR2,       /* Select Status Register 2             */
      SCI_D_SCIDRH,       /* Select Data Register High            */
      SCI_D_SCIDRL,       /* Select Data Register Low             */
      SCI_D_SCIPURD,      /* Select Pullup and Reduced Drive Reg. */
      SCI_D_SCIPORT,      /* Select Port Data Register            */
      SCI_D_SCIDDR        /* Select Data Direction Register       */
} SCI_D_Register_t;
```

16.3.4.3 SCI_D_Control_t – Control Selection

```
typedef enum {
      SCI_D_DISABLE,        /* Disable Selection       */
      SCI_D_ENABLE,         /* Enable Selection        */
      SCI_D_NO_CHANGE       /* Do not change selection */
} SCI_D_Control_t;
```

16.3.4.4 SCI_D_Boolean_t – Boolean Selection

```
typedef enum {
      SCI_D_FALSE,         /* false   */
      SCI_D_TRUE           /* true    */
} SCI_D_Boolean_t;
```

16.3.4.5 SCI_D_Parity_t – SCI Parity Selection

```
typedef enum {
      SCI_D_PARITY_EVEN,   /* Select even parity  */
      SCI_D_PARITY_ODD,    /* Select odd parity   */
      SCI_D_PARITY_NONE    /* Select no parity    */
} SCI_D_Parity_t;
```

16.3.4.6 SCI_D_DataFrame_t – SCI Data Bits Selection

```
typedef enum {
      SCI_D_DATA_8,        /* Select 8 bit data bits */
      SCI_D_DATA_9         /* Select 9 bit data bits */
} SCI_D_DataFrame_t;
```

16.3.4.7 SCI_D_PinMode_t – Pin Mode Selection

```
typdef enum {
      SCI_D_PINMODE_CMOS,         /* CMOS output       */
      SCI_D_PINMODE_OPEN_DRAIN    /* Open-drain output */
} SCI_D_PinMode_t;
```

16.3.4.8 SCI_D_Idle_t - SCI Idle-line Detection

```
typedef enum {
      SCI_D_IDLE_SHORT,     /* Idle character bit count begins after start bit */
      SCI_D_IDLE_LONG       /* Idle character bit count begins after stop bit  */
} SCI_D_Idle_t;
```

16.3.4.9 SCI_D_Wakeup_t - SCI Receiver Wakeup Selection

```
typedef enum {
     SCI_D_WAKEUP_IDLE,     /* SCI receiver awakened by idle-line detection */
     SCI_D_WAKEUP_ADDR      /* SCI receiver awakened by address mark        */
} SCI_D_Wakeup_t;
```

16.3.4.10 SCI_D_RecSource_t – SCI Receiver Source Mode

```
typedef enum {
      SCI_D_RECSOURCE_LOOP,  /* Receiver input internally connected to output */
      SCI_D_RECSOURCE_WIRE   /* Receiver input connected to TXD pin           */
} SCI_D_RecSource_t;
```

16.3.4.11 SCI_D_DataDir_t – Pin Data Direction

```
typedef enum {
      SCI_D_DATADIR_INPUT,      /* Configure pin as input  */
      SCI_D_DATADIR_OUTPUT      /* Configure pin as output */
} SCI_D_DataDir_t;
```

16.3.4.12 SCI_D_PinDrive_t – Pin Drive Capability

```
typedef enum {
      SCI_D_PINDRIVE_FULL,     /* Configure pin for full drive capability    */
      SCI_D_PINDRIVE_REDUCED   /* Configure pin for reduced drive capability */
} SCI_D_PinDrive_t;
```

16.3.4.13 SCI_D_Port_t – SCI Ports

```
typedef enum {
     SCI_D_PORT0,        /* Select Port 0 */
     SCI_D_PORT1,        /* Select Port 1 */
     SCI_D_PORT2,        /* Select Port 2 */
     SCI_D_PORT3,        /* Select Port 3 */
     SCI_D_PORT4,        /* Select Port 4 */
     SCI_D_PORT5,        /* Select Port 5 */
     SCI_D_PORT6,        /* Select Port 6 */
     SCI_D_PORT7,        /* Select Port 7 */
} SCI_D_Port_t;
```

16.3.4.14 SCI_D_PinSense – Pin Sense

```
typedef enum {
     SCI_D_LOW,           /* Active LOW */
     SCI_D_HIGH           /* Active HI  */
} SCI_D_PinSense_t;
```

16.3.5 Register Macros

16.3.5.1 SCI Baud Rate High Register Macros

```
#define SCIBDH_SBR8_BITNO   0
#define SCIBDH_SBR8_MASK   (1 << SCIBDH_SBR8_BITNO)   /* 00000001 */
#define SCIBDH_SBR9_BITNO   1
#define SCIBDH_SBR9_MASK   (1 << SCIBDH_SBR9_BITNO)   /* 00000010 */
#define SCIBDH_SBR10_BITNO   2
#define SCIBDH_SBR10_MASK   (1 << SCIBDH_SBR10_BITNO) /* 00000100 */
#define SCIBDH_SBR11_BITNO   3
#define SCIBDH_SBR11_MASK   (1 << SCIBDH_SBR11_BITNO) /* 00001000 */
#define SCIBDH_SBR12_BITNO   4
#define SCIBDH_SBR12_MASK   (1 << SCIBDH_SBR12_BITNO) /* 00010000 */

#define SCIBDH_SBR_BITNO   0
#define SCIBDH_SBR_MASK   (0x1F)     /* 00011111 */
#define SCIBDH_RESET_MASK   (0x00)   /* 00000000 */
```

16.3.5.2 SCI Baud Rate Low Register Macros

```
#define SCIBDL_SBR0_BITNO   0
#define SCIBDL_SBR0_MASK   (1 << SCIBDH_SBR0_BITNO)   /* 00000001 */
#define SCIBDL_SBR1_BITNO   1
#define SCIBDL_SBR1_MASK   (1 << SCIBDH_SBR1_BITNO)   /* 00000010 */
#define SCIBDL_SBR2_BITNO   2
#define SCIBDL_SBR2_MASK   (1 << SCIBDH_SBR2_BITNO)   /* 00000100 */
#define SCIBDL_SBR3_BITNO   3
#define SCIBDL_SBR3_MASK   (1 << SCIBDH_SBR3_BITNO)   /* 00001000 */
#define SCIBDL_SBR4_BITNO   4
#define SCIBDL_SBR4_MASK   (1 << SCIBDH_SBR4_BITNO)   /* 00010000 */
```

```
#define SCIBDL_SBR5_BITNO   5
#define SCIBDL_SBR5_MASK   (1 << SCIBDH_SBR5_BITNO)   /* 00100000 */
#define SCIBDL_SBR6_BITNO   6
#define SCIBDL_SBR6_MASK   (1 << SCIBDH_SBR6_BITNO)   /* 01000000 */
#define SCIBDL_SBR7_BITNO   7
#define SCIBDL_SBR7_MASK   (1 << SCIBDH_SBR7_BITNO)   /* 10000000 */

#define SCIBDL_SBR_BITNO   0
#define SCIBDL_SBR_MASK   (0xFF)                      /* 11111111 */
#define SCIBDL_RESET_MASK  (1 << SCIBDH_SBR2_BITNO)   /* 00000100 */
```

16.3.5.3 SCI Control Register 1 Macros

```
#define SCICR1_PT_BITNO   0
#define SCICR1_PT_MASK   (1 << SCICR1_PT_BITNO)       /* 00000001 */
#define SCICR1_PE_BITNO   1
#define SCICR1_PE_MASK   (1 << SCICR1_PE_BITNO)       /* 00000010 */
#define SCICR1_ILT_BITNO   2
#define SCICR1_ILT_MASK   (1 << SCICR1_ILT_BITNO)     /* 00000100 */
#define SCICR1_WAKE_BITNO   3
#define SCICR1_WAKE_MASK   (1 << SCICR1_WAKE_BITNO)   /* 00001000 */
#define SCICR1_M_BITNO   4
#define SCICR1_M_MASK   (1 << SCICR1_M_BITNO)         /* 00010000 */
#define SCICR1_RSRC_BITNO   5
#define SCICR1_RSRC_MASK   (1 << SCICR1_RSRC_BITNO)   /* 00100000 */
#define SCICR1_WOMS_BITNO   6
#define SCICR1_WOMS_MASK   (1 << SCICR1_WOMS_BITNO)   /* 01000000 */
#define SCICR1_LOOPS_BITNO   7
#define SCICR1_LOOPS_MASK   (1 << SCICR1_LOOPS_BITNO) /* 10000000 */

#define SCICR1_RESET_MASK   (0x00)          /* 00000000 */
```

16.3.5.4 SCI Control Register 2 Macros

```
#define SCICR2_SBK_BITNO   0
#define SCICR2_SBK_MASK   (1 << SCICR1_SBK_BITNO)   /* 00000001 */
#define SCICR2_RWU_BITNO   1
#define SCICR2_RWU_MASK   (1 << SCICR1_RWU_BITNO)   /* 00000010 */
#define SCICR2_RE_BITNO   2
#define SCICR2_RE_MASK   (1 << SCICR1_RE_BITNO)     /* 00000100 */
#define SCICR2_TE_BITNO   3
#define SCICR2_TE_MASK   (1 << SCICR1_TE_BITNO)     /* 00001000 */
#define SCICR2_ILIE_BITNO   4
#define SCICR2_ILIE_MASK   (1 << SCICR1_ILIE_BITNO) /* 00010000 */
#define SCICR2_RIE_BITNO   5
#define SCICR2_RIE_MASK   (1 << SCICR1_RIE_BITNO)   /* 00100000 */
#define SCICR2_TCIE_BITNO   6
#define SCICR2_TCIE_MASK   (1 << SCICR1_TCIE_BITNO) /* 01000000 */
#define SCICR2_TIE_BITNO   7
#define SCICR2_TIE_MASK   (1 << SCICR1_TIE_BITNO)   /* 10000000 */
```

```
#define SCICR2_RESET_MASK   (0x00)   /* 00000000 */
```

16.3.5.5 SCI Status Register 1 Macros

```
#define SCISR1_PF_BITNO   0
#define SCISR1_PF_MASK   (1 << SCICR1_PF_BITNO)       /* 00000001 */
#define SCISR1_FE_BITNO   1
#define SCISR1_FE_MASK   (1 << SCICR1_FE_BITNO)       /* 00000010 */
#define SCISR1_NF_BITNO   2
#define SCISR1_NF_MASK   (1 << SCICR1_NF_BITNO)       /* 00000100 */
#define SCISR1_OR_BITNO   3
#define SCISR1_OR_MASK   (1 << SCICR1_OR_BITNO)       /* 00001000 */
#define SCISR1_IDLE_BITNO   4
#define SCISR1_IDLE_MASK   (1 << SCICR1_IDLE_BITNO)   /* 00010000 */
#define SCISR1_RDRF_BITNO   5
#define SCISR1_RDRF_MASK   (1 << SCICR1_RDRF_BITNO)   /* 00100000 */
#define SCISR1_TC_BITNO   6
#define SCISR1_TC_MASK   (1 << SCICR1_TC_BITNO)       /* 01000000 */
#define SCISR1_TDRE_BITNO   7
#define SCISR1_TDRE_MASK   (1 << SCICR1_TDRE_BITNO)   /* 10000000 */

#define SCISR1_RESET_MASK   (0x00)      /* 00000000 */
```

16.3.5.6 SCI Status Register 2 Macros

```
#define SCISR2_RAF_BITNO   0
#define SCISR2_RAF_MASK   (1 << SCICR1_RAF_BITNO)   /* 00000001 */

#define SCISR2_RESET_MASK   (0x00)     /* 00000000 */
```

16.3.5.7 SCI Data Register High Macros

```
#define SCIDRH_T8_BITNO   6
#define SCIDRH_T8_MASK   (1 << SCICR1_RAF_BITNO)   /* 01000000 */
#define SCIDRH_R8_BITNO   7
#define SCIDRH_R8_MASK   (1 << SCICR1_RAF_BITNO)   /* 10000000 */

#define SCIDRH_RESET_MASK   (0x00)   /* 00000000 */
```

16.3.5.8 SCI Data Register Low Macros

```
#define SCIDRL_T0_BITNO   0
#define SCIDRL_T0_MASK   (1 << SCICR1_T0_BITNO)   /* 00000001 */
#define SCIDRL_R0_BITNO   0
#define SCIDRL_R0_MASK   (1 << SCICR1_R0_BITNO)   /* 00000001 */
```

```
#define SCIDRL_T1_BITNO   1
#define SCIDRL_T1_MASK   (1 << SCICR1_T1_BITNO)   /* 00000010 */
#define SCIDRL_R1_BITNO   1
#define SCIDRL_R1_MASK   (1 << SCICR1_R1_BITNO)   /* 00000010 */
#define SCIDRL_T2_BITNO   2
#define SCIDRL_T2_MASK   (1 << SCICR1_T2_BITNO)   /* 00000100 */
#define SCIDRL_R2_BITNO   2
#define SCIDRL_R2_MASK   (1 << SCICR1_R2_BITNO)   /* 00000100 */
#define SCIDRL_T3_BITNO   3
#define SCIDRL_T3_MASK   (1 << SCICR1_T3_BITNO)   /* 00001000 */
#define SCIDRL_R3_BITNO   3
#define SCIDRL_R3_MASK   (1 << SCICR1_R3_BITNO)   /* 00001000 */
#define SCIDRL_T4_BITNO   4
#define SCIDRL_T4_MASK   (1 << SCICR1_T4_BITNO)   /* 00010000 */
#define SCIDRL_R4_BITNO   4
#define SCIDRL_R4_MASK   (1 << SCICR1_R4_BITNO)   /* 00010000 */
#define SCIDRL_T5_BITNO   5
#define SCIDRL_T5_MASK   (1 << SCICR1_T5_BITNO)   /* 00100000 */
#define SCIDRL_R5_BITNO   5
#define SCIDRL_R5_MASK   (1 << SCICR1_R5_BITNO)   /* 00100000 */
#define SCIDRL_T6_BITNO   6
#define SCIDRL_T6_MASK   (1 << SCICR1_T6_BITNO)   /* 01000000 */
#define SCIDRL_R6_BITNO   6
#define SCIDRL_R6_MASK   (1 << SCICR1_R6_BITNO)   /* 01000000 */
#define SCIDRL_T7_BITNO   7
#define SCIDRL_T7_MASK   (1 << SCICR1_T7_BITNO)   /* 10000000 */
#define SCIDRL_R7_BITNO   7
#define SCIDRL_R7_MASK   (1 << SCICR1_R7_BITNO)   /* 10000000 */

#define SCIDRL_T_MASK   (0xFF)    /* 11111111 */
#define SCIDRL_R_MASK   (0xFF)    /* 11111111 */

#define SCIDRL_RESET_MASK   (0x00)   /* 00000000 */
```

16.3.5.9 SCI Pullup and Reduced Drive Register Macros

```
#define SCIPURD_PUPSCI0_BITNO   0
#define SCIPURD_PUPSCI0_MASK   (1 << SCICR1_PUPSCI0_BITNO)   /* 00000001 */
#define SCIPURD_PUPSCI1_BITNO   1
#define SCIPURD_PUPSCI1_MASK   (1 << SCICR1_PUPSCI1_BITNO)   /* 00000010 */
#define SCIPURD_RDPSCI0_BITNO   4
#define SCIPURD_RDPSCI0_MASK   (1 << SCICR1_RDPSCI0_BITNO)   /* 00010000 */
#define SCIPURD_PUPSCI1_BITNO   5
#define SCIPURD_PUPSCI1_MASK   (1 << SCICR1_RDPSCI1_BITNO)   /* 00100000 */
#define SCIPURD_SCISDOZ_BITNO   7
#define SCIPURD_SCISDOZ_MASK   (1 << SCICR1_PUPSCI_BITNO)    /* 10000000 */

#define SCIPURD_RESET_MASK   (0x00)    /* 00000000 */
```

16.3.5.10 SCI Port Data Register Macros

```
#define SCIPORT_PORT0_BITNO   0
#define SCIPORT_PORT0_MASK   (1 << SCIPORT_PORT0_BITNO)   /* 00000001 */
#define SCIPORT_PORT1_BITNO   1
#define SCIPORT_PORT1_MASK   (1 << SCIPORT_PORT1_BITNO)   /* 00000010 */
#define SCIPORT_PORT2_BITNO   2
#define SCIPORT_PORT2_MASK   (1 << SCIPORT_PORT2_BITNO)   /* 00000100 */
#define SCIPORT_PORT3_BITNO   3
#define SCIPORT_PORT3_MASK   (1 << SCIPORT_PORT3_BITNO)   /* 00001000 */
#define SCIPORT_PORT4_BITNO   4
#define SCIPORT_PORT4_MASK   (1 << SCIPORT_PORT4_BITNO)   /* 00010000 */
#define SCIPORT_PORT5_BITNO   5
#define SCIPORT_PORT5_MASK   (1 << SCIPORT_PORT5_BITNO)   /* 00100000 */
#define SCIPORT_PORT6_BITNO   6
#define SCIPORT_PORT6_MASK   (1 << SCIPORT_PORT6_BITNO)   /* 01000000 */
#define SCIPORT_PORT7_BITNO   7
#define SCIPORT_PORT7_MASK   (1 << SCIPORT_PORT7_BITNO)   /* 10000000 */

#define SCIPORT_PORT_MASK   (0xFF)   /* 11111111 */
#define SCIPORT_RESET_MASK   (0x00)   /* 00000000 */
```

16.3.5.11 SCI Data Direction Register Macros

```
#define SCIDDR_DDR0_BITNO   0
#define SCIDDR_DDR0_MASK   (1 << SCIDDR_DDR0_BITNO)   /* 00000001 */
#define SCIDDR_DDR1_BITNO   1
#define SCIDDR_DDR1_MASK   (1 << SCIDDR_DDR1_BITNO)   /* 00000010 */
#define SCIDDR_DDR2_BITNO   2
#define SCIDDR_DDR2_MASK   (1 << SCIDDR_DDR2_BITNO)   /* 00000100 */
#define SCIDDR_DDR3_BITNO   3
#define SCIDDR_DDR3_MASK   (1 << SCIDDR_DDR3_BITNO)   /* 00001000 */
#define SCIDDR_DDR4_BITNO   4
#define SCIDDR_DDR4_MASK   (1 << SCIDDR_DDR4_BITNO)   /* 00010000 */
#define SCIDDR_DDR5_BITNO   5
#define SCIDDR_DDR5_MASK   (1 << SCIDDR_DDR5_BITNO)   /* 00100000 */
#define SCIDDR_DDR6_BITNO   6
#define SCIDDR_DDR6_MASK   (1 << SCIDDR_DDR6_BITNO)   /* 01000000 */
#define SCIDDR_DDR7_BITNO   7
#define SCIDDR_DDR7_MASK   (1 << SCIDDR_DDR7_BITNO)   /* 10000000 */

#define SCIDDR_DDR_MASK   (0xFF)     /* 11111111 */
#define SCIDDR_RESET_MASK   (0x00)   /* 00000000 */
```

16.3.6 Miscellaneous Macros

```
#define MAX_7BIT_TRANSMIT_VALUE  127
#define MAX_8BIT_TRANSMIT_VALUE  255
#define MAX_9BIT_TRANSMIT_VALUE  511
```

16.3.7 Return Codes

Return codes from the enumerated type “SCI_D_ReturnCode_t”.

Return Code	Functions Returning	Description
SCI_D_ERR_NONE	All	No error.
SCI_D_ERR_INVALID_HANDLE	All	SCI base address parameter is zero.
SCI_D_ERR_BAD_RESULT_ADDR	Receive, GetRegister, ReadPin, Transmit, GetStatus	Result Pointer is zero.
SCI_D_ERR_INVALID_ADDR	ConfigureGPIO	Pointer is zero.
SCI_D_ERR_INVALID_REGISTER	SetRegister, GetRegister	SCI Register Selection switch is invalid.
SCI_D_ERR_INVALID_CONTROL	Init, ControlOperation, ControlInterrupt, ControlLoopback, ConfigureGPIO	Control choice is not valid.
SCI_D_ERR_INVALID_BAUD_RATE	Init, SetBaudRate	Baud rate is not valid.
SCI_D_ERR_INVALID_CLOCK_VALUE	Init, SetBaudRate	SCI Clock rate value is invalid.
SCI_D_ERR_INVALID_CLOCK_DIVIDER	Init, SetBaudRate	Baud rate and/or SCI clock rate values were invalid and lead to invalid clock divider value
SCI_D_ERR_INVALID_PIN_MODE	Init, ControlLoopback	Pin mode is invalid (CMOS or open-drain).
SCI_D_ERR_INVALID_IDLE_MODE	Init	Idle-line detect type is invalid.
SCI_D_ERR_INVALID_PARITY	Init	Parity value is invalid.
SCI_D_ERR_INVALID_DATA_FRAME	Init	Data frame value is invalid.
SCI_D_ERR_INVALID_WAKEUP_MODE	Init	Wakeup mode is invalid.
SCI_D_ERR_INVALID_DATA_VALUE	Transmit	Data value is invalid.
SCI_D_ERR_INVALID_RECEIVER_SOURCE	ControlLoopback	Receiver source is invalid.
SCI_D_ERR_INVALID_DATA_DIRECTION	ControlLoopback, ConfigureGPIO	Port data direction (output or input) is invalid.
SCI_D_ERR_INVALID_DRIVE_MODE	ConfigureGPIO	Reduced drive mode is invalid.
SCI_D_ERR_INVALID_PORT_NUMBER	ReadPin, WritePin	Port Pin number is invalid.
SCI_D_ERR_INVALID_PIN_SENSE	WritePin	Pin Sense (high or low) is invalid.
SCI_D_ERR_WAKEUP_PARITY_CONFLICT	Init	Address mark wakeup and parity are enabled

16.4 SCI_D API Definitions

16.4.1 SCI_D_Init

16.4.1.1 Description

The SCI_D_Init function initializes the SCI. This function should be called once during the system software initialization or for reinitialization of the module. Parameter checking is a compile-time option.

A note about the PUPSCI1 bit and the PUPSCI2 bit of the register SCIPURD: according to the Serial Communications Interface User Guide, Revision 1.0.2, the reset value for these bits is determined at SoC level; however, according to section 2.2.4 of the SIKA Overview Specification, Revision 1.2, the default reset values for these bits is zero, and the pull-up function is disabled by default. The SCI_D device driver assumes that the reset value for these bits is clear.

16.4.1.2 Prototype

```
SCI_D_ReturnCode_t   SCI_D_Init (
         pSCI_D_t           SCIPtr,
         UINT32             baudRateValue,
         UINT32             SCIClockValue,
         SCI_D_PinMode_t    pinModeValue,
         SCI_D_Idle_t       idleValue,
         SCI_D_Parity_t     parityValue,
         SCI_D_DataFrame_t  dataFrameValue,
         SCI_D_Wakeup_t     wakeUpValue,
         SCI_D_Control_t    dozeModeControl
)
```

16.4.1.3 Arguments

Data Type	Formal Name	Description	Valid Values
pSCI_D_t	SCIPtr	SCI base address associated with this driver.	Any non-zero core processor data address.
UINT32	baudRateValue	Desired baud rate (in bits per second)	Any non-zero 32-bit value
UINT32	SCIClockValue	SCI module clock rate (in Hz)	Any non-zero 32-bit value
SCI_D_PinMode_t	pinModeValue	Output mode	SCI_D_CMOS, SCI_D_OPEN_DRAIN
SCI_D_Idle_t	idleValue	Idle-line detect mode	SCI_D_IDLE_SHORT, SCI_D_IDLE_LONG
SCI_D_Parity_t	parityValue	Parity Mode	SCI_D_PARITY_NONE, SCI_D_PARITY_ODD, SCI_D_PARITY_EVEN
SCI_D_DataFrame_t	dataFrameValue	Data Frame Length	SCI_D_DATA_8, SCI_D_DATA_9
SCI_D_Wakeup_t	wakeUpValue	Wakeup type	SCI_D_WAKEUP_ADDR, SCI_D_WAKEUP_IDLE
SCI_D_Control_t	dozeModeControl	Enable or disable SCI when board is in doze mode	SCI_D_DISABLE, SCI_D_ENABLE

Note that an invalid state will occur if the argument wakeUpValue is set to SCI_D_WAKEUP_ADDR and the argument parityValue is set to either SCI_D_PARITY_ODD or SCI_D_PARITY_EVEN. If the user supplies this configuration of parameters when calling the SCI_D_Init function, the return code SCI_D_ERR_WAKEUP_PARITY_CONFLICT will be returned.

16.4.1.4 Return Values

Error Code	Description
SCI_D_ERR_NONE	No error
SCI_D_ERR_INVALID_HANDLE	SCI base address parameter is zero.
SCI_D_ERR_INVALID_CONTROL	Not a valid control selection
SCI_D_ERR_INVALID_BAUD_RATE	Baud rate is not valid. (baud rate less than zero)
SCI_D_ERR_INVALID_CLOCK_VALUE	SCI Clock rate value is invalid. (clock rate less than zero)
SCI_D_ERR_INVALID_CLOCK_DIVIDER	Baud rate and/or SCI clock rate values were invalid and lead to invalid clock divider value
SCI_D_ERR_INVALID_PIN_MODE	Pin mode value is invalid
SCI_D_ERR_INVALID_IDLE_MODE	Idle-line detect type is invalid
SCI_D_ERR_INVALID_PARITY	Parity value is invalid
SCI_D_ERR_INVALID_DATA_FRAME	Data frame value is invalid
SCI_D_ERR_INVALID_WAKEUP_MODE	Wakeup mode is invalid
SCI_D_ERR_WAKEUP_PARITY_CONFLICT	Address mark wakeup and parity are enabled

16.4.2 SCI_D_ControlOperation

16.4.2.1 Description

The SCI_D_ControlOperation function enables, disables, or leaves unchanged the SCI transmitter and the SCI receiver. Parameter checking is a compile-time option.

This function should only be called after the SCI has been initialized through a previous call to the SCI_D_Init function.

16.4.2.2 Prototype

```
SCI_D_ReturnCode_t   SCI_D_ControlOperation (
         pSCI_D_t          SCIPtr,
         SCI_DControl_t    controlTransmitter,
         SCI_DControl_t    controlReceiver
)
```

16.4.2.3 Arguments

Data Type	Formal Name	Description	Valid Values
pSCI_D_t	SCIPtr	SCI base address associated with this driver.	Any non-zero core processor data address.
SCI_D_Control_t	controlTransmitter	Control Transmitter enabling	SCI_D_DISABLE, SCI_D_ENABLE, SCI_D_NO_CHANGE
SCI_D_Control_t	controlReceiver	Control Receiver enabling	SCI_D_DISABLE, SCI_D_ENABLE, SCI_D_NO_CHANGE

16.4.2.4 Return Values

Error Code	Description
SCI_D_ERR_NONE	No error
SCI_D_ERR_INVALID_HANDLE	SCI base address parameter is zero
SCI_D_ERR_INVALID_CONTROL	Not a valid control selection

16.4.3 SCI_D_ControlInterrupt

16.4.3.1 Description

The SCI_D_ControlInterrupt function enables and disables SCI interrupts. Parameter checking is a compile-time option.

This function should only be called after the SCI has been initialized through a previous call to SCI_D_Init function.

16.4.3.2 Prototype

```
SCI_D_ReturnCode_t   SCI_D_ControlInterrupt (
         pSCI_D_t           SCIPtr,
         SCI_D_Control_t    transmitterInt,
         SCI_D_Control_t    transCompleteInt,
         SCI_D_Control_t    receiverInt,
         SCI_D_Control_t    idleLineInt
)
```

16.4.3.3 Arguments

Data Type	Formal Name	Description	Valid Values
pSCI_D_t	SCIPtr	SCI base address associated with this driver.	Any non-zero core processor data address.
SCI_D_Control_t	transmitInt	Control transmit interrupts	SCI_D_DISABLE, SCI_D_ENABLE, SCI_D_NO_CHANGE
SCI_D_Control_t	transCompleteInt	Control transmit complete interrupts	SCI_D_DISABLE, SCI_D_ENABLE, SCI_D_NO_CHANGE
SCI_D_Control_t	receiverInt	Control receiver interrupts	SCI_D_DISABLE, SCI_D_ENABLE, SCI_D_NO_CHANGE
SCI_D_Control_t	idleLineInt	Control idle line interrupts	SCI_D_DISABLE, SCI_D_ENABLE, SCI_D_NO_CHANGE

16.4.3.4 Return Values

Error Code	Description
SCI_D_ERR_NONE	No error
SCI_D_ERR_INVALID_HANDLE	SCI base address parameter is zero
SCI_D_ERR_INVALID_CONTROL	Not a valid control selection

16.4.4 SCI_D_Transmit

16.4.4.1 Description

The SCI_D_Transmit function writes data to the SCI data registers SCIDRL and SCIDRH and returns the transmitter status. Parameter checking is a compile time option.

16.4.4.2 Prototype

```
SCI_D_ReturnCode_t   SCI_D_Transmit (
         pSCI_D_t             SCIPtr,
         UINT16               dataVal,
         SCI_D_Boolean_t *    transmitDataPending
)
```

16.4.4.3 Arguments

Data Type	Formal Name	Description	Valid Values
pSCI_D_t	SCIPtr	SCI base address associated with this driver.	Any non-zero core processor data address.
UINT16	DataVal	Data value to transmit	0-MAX_7BIT_TRANSMIT_VALUE if 8-bit data frame with parity enabled 0-MAX_8BIT_TRANSMIT_VALUE if 8-bit data frame with parity disabled 0-MAX_8BIT_TRANSMIT_VALUE if 9-bit data frame with parity enabled 0-MAX_9BIT_TRANSMIT_VALUE if 9-bit data frame with parity disabled
SCI_D_Boolean_t *	transmitDataPending	Result address for transmission status	Any non-zero core processor data address.

16.4.4.4 Return Values

Error Code	Description
SCI_D_ERR_NONE	No error
SCI_D_ERR_INVALID_HANDLE	SCI base address parameter is zero
SCI_D_ERR_INVALID_DATA_VALUE	Data value is invalid with mode select value
SCI_D_ERR_BAD_RESULT_ADDR	Result Pointer is zero.

16.4.5 SCI_D_Receive

16.4.5.1 Description

The SCI_D_Receive function returns the contents of the SCI data registers SCIDRL and SCIDRH and returns the receiver status. Parameter checking is a compile-time option.

This function should only be called after the SCI has been initialized through a previous call to SCI_D_Init.

16.4.5.2 Prototype

```
SCI_D_ReturnCode_t   SCI_D_Receive (
         pSCI_D_t                    SCIPtr,
         UINT16                     *dataPtr,
         pSCI_D_ReceiverStatus _t    StatusPtr
)
```

16.4.5.3 Arguments

Data Type	Formal Name	Description	Valid Values
pSCI_D_t	SCIPtr	SCI_D base address associated with this driver.	Any non-zero core processor data address.
UINT16 *	dataPtr	Return address for received data.	Any non-zero core processor data address.
pSCI_D_ReceiverStatus _t	StatusPtr	Structure for status fields	Any non-zero core processor data address.

16.4.5.4 Return Values

Error Code	Description
SCI_D_ERR_NONE	No error
SCI_D_ERR_INVALID_HANDLE	SCI base address parameter is zero
SCI_D_ERR_BAD_RESULT_ADDR	Result pointer is zero

16.4.6 SCI_D_ControlBreak

16.4.6.1 Description

The SCI_D_ControlBreak function enables or disables the sending of break characters. Parameter checking is a compile-time option.

This function should only be called after the SCI has been initialized through a previous call to SCI_D_Init function. Ideally, the SCI_D_ControlBreak function should be called to enable break character transmission, a user-defined delay should occur, and then the SCI_D_ControlBreak function should be called again to disable break character transmission.

16.4.6.2 Prototype

```
SCI_D_ReturnCode_t   SCI_D_ControlBreak (
         pSCI_D_t           SCIPtr,
         SCI_D_Control_t    controlBreak
)
```

16.4.6.3 Arguments

Data Type	Formal Name	Description	Valid Values
pSCI_D_t	SCIPtr	SCI_D base address associated with this driver.	Any non-zero core processor data address.
SCI_D_Control_t	controlBreak	Enables or disables the sending of break characters	SCI_D_ENABLE, SCI_D_DISABLE

16.4.6.4 Return Values

Error Code	Description
SCI_D_ERR_NONE	No error
SCI_D_ERR_INVALID_HANDLE	SCI base address parameter is zero
SCI_D_ERR_INVALID_CONTROL	Not a valid control selection

16.4.7 SCI_D_ControlLoopback

16.4.7.1 Description

The SCI_D_ControlLoopback function enables or disables the feedback path on the data serial shifter and sets the mode of loop operation based upon user input. Parameter checking is a compile-time option.

This function should only be called after the SCI has been initialized through a previous call to SCI_D_Init.

16.4.7.2 Prototype

```
SCI_D_ReturnCode_t   SCI_D_ControlLoopback (
         pSCI_D_t             SCIPtr,
         SCI_D_Control_t      controlLoopback,
         SCI_D_RecSource_t    recSourceValue,
         SCI_D_DataDir_t      transDirValue,
         SCI_D_DataDir_t      recDirValue,
         SCI_D_PinMode_t      pinModeValue
)
```

16.4.7.3 Arguments

Data Type	Formal Name	Description	Valid Values
pSCI_D_t	SCIPtr	SCI_D base address associated with this driver.	Any non-zero core processor data address.
SCI_D_Control_t	controlLoopback	Control loopback mode	SCI_D_DISABLE, SCI_D_ENABLE, SCI_D_NO_CHANGE
SCI_D_RecSource_t	recSourceValue	Receiver source type (loop or single wire)	SCI_D_RECSOURCE_WIRE, SCI_D_RECSOURCE_LOOP
SCI_D_DataDir_t	transDirValue	Transmitter direction value (configure as output or input)	SCI_D_DATADIR_OUTPUT, SCI_D_DATADIR_INPUT
SCI_D_DataDir_t	recDirValue	Receiver direction value (configure as output or input)	SCI_D_DATADIR_OUTPUT, SCI_D_DATADIR_INPUT
SCI_D_PinMode_t	pinModeValue	Pin mode value (CMOS or open drain)	SCI_D_PINMODE_CMOS, SCI_D_PINMODE_OPEN_DRAIN

16.4.7.4 Return Values

Error Code	Description
SCI_D_ERR_NONE	No error
SCI_D_ERR_INVALID_HANDLE	SCI base address parameter is zero
SCI_D_ERR_INVALID_CONTROL	Control choice is not valid
SCI_D_ERR_INVALID_RECEIVER_SOURCE	Receiver source is invalid
SCI_D_ERR_INVALID_DATA_DIRECTION	Port data direction is invalid
SCI_D_ERR_INVALID_PIN_MODE	Pin mode is invalid

16.4.8 SCI_D_SetBaudRate

16.4.8.1 Description

The SCI_D_SetBaudRate function sets the baud rate of the SCI. Parameter checking is a compile-time option.

This function should only be called after the SCI has been initialized through a previous call to SCI_D_Init function.

16.4.8.2 Prototype

```
SCI_D_ReturnCode_t   SCI_D_SetBaudRate (
         pSCI_D_t    SCIPtr,
         UINT32      baudRateValue,
         UINT32      SCIClockValue
)
```

16.4.8.3 Arguments

Data Type	Formal Name	Description	Valid Values
pSCI_D_t	SCIPtr	SCI_D base address associated with this driver.	Any non-zero core processor data address.
UINT32	baudRateValue	Desired baud rate (in bits per second)	Any non-zero 32-bit value
UINT32	SCIClockValue	SCI module clock rate (in Hz)	Any non-zero 32-bit value

16.4.8.4 Return Values

Error Code	Description
SCI_D_ERR_NONE	No error
SCI_D_ERR_INVALID_HANDLE	SCI base address parameter is zero
SCI_D_ERR_INVALID_BAUD_RATE	Baud rate is not valid. (baud rate is zero)
SCI_D_ERR_INVALID_CLOCK_VALUE	SCI Clock rate value is invalid. (clock rate is zero)
SCI_D_ERR_INVALID_CLOCK_DIVIDER	Baud rate and/or SCI clock rate values were invalid and lead to invalid clock divider value

16.4.9 SCI_D_SetRegister

16.4.9.1 Description

The SCI_D_SetRegister function writes an 8-bit value to one of the SCI registers. Parameter checking is a compile-time option.

16.4.9.2 Prototype

```
SCI_D_ReturnCode_t  SCI_D_SetRegister (
         pSCI_D_t            SCIPtr,
         SCI_D_Register_t    registerSwitch,
         UINT8               registerValue
)
```

16.4.9.3 Arguments

Data Type	Formal Name	Description	Valid Values
pSCI_D_t	SCIPtr	SCI_D base address associated with this driver.	Any non-zero core processor data address.
SCI_D_Register_t	registerSwitch	Select among SCI_D registers	SCI_D_SCIBDH, SCI_D_SCIBDL, SCI_D_SCICR1, SCI_D_SCICR2, SCI_D_SCIDRH, SCI_D_SCIDRL, SCI_D_SCIPURD, SCI_D_SCIPORT, SCI_D_SCIDDR
UINT8	registerValue	Data to copy into selected register	0, 1, 2, …, 255

16.4.9.4 Return Values

Error Code	Description
SCI_D_ERR_NONE	No error
SCI_D_ERR_INVALID_HANDLE	SCI base address parameter is zero
SCI_D_ERR_INVALID_REGISTER	SCI Register Selection switch is invalid

16.4.10 SCI_D_GetRegister

16.4.10.1 Description

The SCI_D_GetRegister function returns the contents of one of the SCI registers. Parameter checking is a compile-time option.

This function can be called at any time.

16.4.10.2 Prototype

```
SCI_D_ReturnCode_t   SCI_D_GetRegister (
          pSCI_D_t             SCIPtr,
          SCI_D_Register_t     registerSwitch,
          UINT8               *getRegisterPtr
)
```

16.4.10.3 Arguments

Data Type	Formal Name	Description	Valid Values
pSCI_D_t	SCIPtr	SCI_D base address associated with this driver.	Any non-zero core processor data address.
SCI_D_Register_t	registerSwitch	Select among SCI_D registers	SCI_D_SCIBDH, SCI_D_SCIBDL, SCI_D_SCICR1, SCI_D_SCICR2, SCI_D_SCISR1, SCI_D_SCISR2, SCI_D_SCIDRH, SCI_D_SCIDRL, SCI_D_SCIPURD, SCI_D_SCIPORT, SCI_D_SCIDDR
UINT8 *	getRegisterPtr	Result address for selected SCI_D register	Any non-zero core processor data address.

16.4.10.4 Return Values

Error Code	Description
SCI_D_ERR_NONE	No error
SCI_D_ERR_INVALID_HANDLE	SCI base address parameter is zero
SCI_D_ERR_INVALID_REGISTER	SCI Register Selection switch is invalid
SCI_D_ERR_BAD_RESULT_ADDR	Result pointer is zero

16.4.11 SCI_D_ConfigureGPIO

16.4.11.1 Description

The SCI_D_ConfigureGPIO function sets data direction for general purpose I/O (GPIO) pins, sets reduced drive status for GPIO pins, and enables, disables, or leaves unchanged pullups for GPIO pins. Parameter checking is a compile-time option.

This function should only be called after the SCI has been initialized through a previous call to SCI_D_Init function.

16.4.11.2 Prototype

```
SCI_D_ReturnCode_t   SCI_D_ConfigureGPIO (
         pSCI_D_t                SCIPtr,
         SCI_D_DataDirStruct_t   dataDir,
         SCI_D_PinDrive_t        pinDrive0_1,
         SCI_D_PinDrive_t        pinDrive2_7,
         SCI_D_Control_t         controlPullups0_1,
         SCI_D_Control_t         controlPullups2_7
)
```

16.4.11.3 Arguments

Data Type	Formal Name	Description	Valid Values
pSCI_D_t	SCIPtr	SCI_D base address associated with this driver.	Any non-zero core processor data address.
SCI_D_DataDirStruct_t	dataDir	Structure that holds Pins data direction	Any non-zero core processor data address.
SCI_D_PinDrive_t	pinDrive0_1	Configure pins 1 and 0 for reduced or full drive capability	SCI_D_PINDRIVE_REDUCED, SCI_D_PINDRIVE_FULL
SCI_D_PinDrive_t	pinDrive2_7	Configure pins 2 through 7 for reduced or full drive capability	SCI_D_PINDRIVE_REDUCED, SCI_D_PINDRIVE_FULL
SCI_D_Control_t	controlPullups0_1	Enable or disable pullups on pins 1 and 0	SCI_D_DISABLE, SCI_D_ENABLE, SCI_D_NO_CHANGE
SCI_D_Control_t	ControlPullups2_7	Enable or disable pullups on pins 2 through 7	SCI_D_DISABLE, SCI_D_ENABLE, SCI_D_NO_CHANGE

16.4.11.4 Return Values

Error Code	Description
SCI_D_ERR_NONE	No error
SCI_D_ERR_INVALID_HANDLE	SCI base address parameter is zero
SCI_D_ERR_INVALID_DATA_DIRECTION	Port data direction is invalid
SCI_D_ERR_INVALID_DRIVE_MODE	Reduced drive mode is invalid
SCI_D_ERR_INVALID_CONTROL	Control choice is invalid
SCI_D_ERR_INVALID_ADDR	Pointer is zero.

16.4.12 SCI_D_ReadPin

16.4.12.1 Description

The SCI_D_ReadPin function returns GPIO pin states. Parameter checking is a compile-time option.

This function should only be called after the SCI has been initialized through a previous call to SCI_D_Init, and GPIO pins have been configured through a previous call to SCI_D_ConfigureGPIO.

16.4.12.2 Prototype

```
SCI_D_ReturnCode_t   SCI_D_ReadPin (
          pSCI_D_t            SCIPtr,
          SCI_D_Port_t        portSwitch,
          SCI_D_PinSense_t  * portValuePtr
)
```

16.4.12.3 Arguments

Data Type	Formal Name	Description	Valid Values
pSCI_D_t	SCIPtr	SCI_D base address associated with this driver.	Any non-zero core processor data address.
SCI_D_Port_t	portSwitch	Select SCI port	SCI_D_PORT0, SCI_D_PORT1, SCI_D_PORT2, SCI_D_PORT3, SCI_D_PORT4, SCI_D_PORT5, SCI_D_PORT6, SCI_D_PORT7
SCI_D_PinSense_t *	portValuePtr	Result address for selected SCI_D port	Any non-zero core processor data address.

16.4.12.4 Return Values

Error Code	Description
SCI_D_ERR_NONE	No error
SCI_D_ERR_INVALID_HANDLE	SCI base address parameter is zero
SCI_D_ERR_INVALID_PORT_NUMBER	Port pin number is invalid
SCI_D_ERR_BAD_RESULT_ADDR	Result pointer is zero

16.4.13 SCI_D_WritePin

16.4.13.1 Description

The SCI_D_WritePin function changes GPIO pin states. Parameter checking is a compile-time option.

16.4.13.2 Prototype

```
SCI_D_ReturnCode_t   SCI_D_WritePin (
          pSCI_D_t            SCIPtr,
          SCI_D_Port_t        portSwitch,
          SCI_D_PinSense_t    portValue
)
```

16.4.13.3 Arguments

Data Type	Formal Name	Description	Valid Values
pSCI_D_t	SCIPtr	SCI_D base address associated with this driver.	Any non-zero core processor data address.
SCI_D_Port_t	portSwitch	Select SCI port	SCI_D_PORT0, SCI_D_PORT1, SCI_D_PORT2, SCI_D_PORT3, SCI_D_PORT4, SCI_D_PORT5, SCI_D_PORT6, SCI_D_PORT7
SCI_D_PinSense_t	portValue	Value to write to SCI port pin	SCI_D_LOW, SCI_D_HIGH

16.4.13.4 Return Values

Error Code	Description
SCI_D_ERR_NONE	No error
SCI_D_ERR_INVALID_HANDLE	SCI base address parameter is zero
SCI_D_ERR_INVALID_PORT_NUMBER	Port pin number is invalid
SCI_D_ERR_INVALID_PIN_SENSE	Pin Sense (high or low) is invalid

16.4.14 SCI_D_Reset

16.4.14.1 Description

The SCI_D_Reset function returns all SCI registers to their reset (power on) values. Parameter checking is a compile-time option.

This function can be called at any time.

Note, the PUPSCI1 and PUPSCI2 bits of register SCIPURD: According to the Serial Communications Interface User Guide, Revision 1.0.2, the reset value for these bits is determined at SoC level; however, according to section 2.2.4 of the SIKA Overview Specification, Revision 1.2, the default reset values for these bits is zero, and the pull-up function is disabled by default. Therefore, SCI_D will assume the reset value for these bits is clear.

16.4.14.2 Prototype

```
SCI_D_ReturnCode_t  SCI_D_Reset (
         pSCI_D_t    SCIPtr
)
```

16.4.14.3 Arguments

Data Type	Formal Name	Description	Valid Values
pSCI_D_t	SCIPtr	SCI_D base address associated with this driver.	Any non-zero core processor data address.

16.4.14.4 Return Values

Error Code	Description
SCI_D_ERR_NONE	No error
SCI_D_ERR_INVALID_HANDLE	SCI base address parameter is zero

16.4.15 SCI_D_GetStatus

16.4.15.1 Description

The SCI_D_GetStatus function returns the SCI status. Parameter checking is a compile-time option.

This function can be called at any time.

16.4.15.2 Prototype

```
SCI_D_ReturnCode_t   SCI_D_GetStatus (
         pSCI_D_t          SCIPtr,
         pSCI_D_Status_t   StatusPtr
)
```

16.4.15.3 Arguments

Data Type	Formal Name	Description	Valid Values
pSCI_D_t	SCIPtr	SCI_D base address associated with this driver.	Any non-zero core processor data address.
pSCI_D_Status_t	StatusPtr	Structure for status fields	Any non-zero core processor data address.

16.4.15.4 Return Values

Error Code	Description
SCI_D_ERR_NONE	No error
SCI_D_ERR_INVALID_HANDLE	SCI base address parameter is zero
SCI_D_BAD_RESULT_ADDR	Result Pointer is zero.

16.5 SCI_D Example Application

The following describes the SCI_D example application.

16.5.1 Description for Example Application

The example application echoes a string of text entered into a terminal or terminal emulator program. It starts by initalizing the SCI, and enabling the transmitter and receiver. The program shall echo characters received, and when a carriage return is detected the program shall transmit the entire preceding line. The program shall continue running until an exit sequence is received.

16.5.1.1 Example Application Code

```
/*******************************************************************

File:    scid_app.c

Purpose: The following example code illustrates the use of the M-CORE
         Peripherals Library SCI_D Level 1 driver.  It is setup to
         operation on an M-CORE 2107 CMB board.

         The program initializes the SCI and waits for terminal input.
         It echoes each character received, and when a carriage return
         is detected, the program transmits the entire preceeding line.

         This program will continue running until either an escape
         character or end-of-transmission (control-D) is received.

         If an error occurs, the program will exit with the line
         number of where the failure occurred.

Time to run:
         About 30 seconds.

General Setup:
         General hardware and software setup to run this application
         can be found in the 'readme_app.txt' file, located in the
         '<library>/app' directory.

Application Specific Setup:
         Development Board: MMC2107 CMB or EVB

(C) Copyright Motorola Inc, 2000.  All rights reserved.
*******************************************************************/

#include <stdlib.h>
#include <stdarg.h>
#include <stdio.h>
#include "sci_d.h"

/*******************************************************************
```

```
   Macro Definitions
/*********************************************************************/

/* macro to handle error checking and response */
#define HANDLE_ERROR(err)  if (err != SCI_D_ERR_NONE) return(__LINE__);

#define EOT 4      /* end-of-transmission character */
#define ESC 033    /* escape character              */
#define STRBUF 255 /* size of input string buffer   */
#define EOS '\0'   /* end of string                 */

/*********************************************************************
   Function Prototypes
*********************************************************************/
int main(void);                                 /* function prototype */
int scid_example(pSCI_D_t);                     /* function prototype */
static void out_string (pSCI_D_t, char *);      /* function prototype */

/*********************************************************************
   Name: main
   Description: Drive program operation
   Inputs: none
   Outputs: status (if passed, 0; else line number of the failure)
*********************************************************************/
int main ()
{
    int status;

    status = scid_example ((pSCI_D_t)__MMC2107_SCI1);

    return (status);
}

/*********************************************************************
   Name: scid_example
   Description: Provide an example for the usage of the SCI_D
   Inputs:  SCIptr    (pointer to SCI_D base address)
   Outputs: status (if passed, 0; else line number of the failure)
*********************************************************************/
int scid_example (pSCI_D_t SCIPtr)
{

 /*-----------------------------------------------------------------*/
 /* declare local variables                                         */
 /*-----------------------------------------------------------------*/

  /* return codes */
  SCI_D_ReturnCode_t status = SCI_D_ERR_NONE; /* return value       */

  /* for reception */
  UINT16 receiveVal;                        /* value to receive     */
  UINT16 * receivePtr = &receiveVal;        /* pointer to receiveVal */
  SCI_D_ReceiverStatus_t statusVal;
  pSCI_D_ReceiverStatus_t statusPtr = &statusVal;/* status structure */
```

```
 /* for transmission */
 SCI_D_Boolean_t transmitDataPending;      /* status boolean        */
 SCI_D_Boolean_t * transmitDataPendingPtr = &transmitDataPending;

 /* line buffer */
 static char lineBuffer[STRBUF];           /* line buffer           */
 char * lineBufferPtr = lineBuffer;        /* pointer to line buffer*/

 SCI_D_Boolean_t readyFlag = SCI_D_FALSE;  /* flag for while loop   */

/*-------------------------------------------------------------------*/
/* reset the SCI                                                     */
/*-------------------------------------------------------------------*/

 /* Call SCI_D_Reset with valid parameters */
 status = SCI_D_Reset( SCIPtr );

 /* If an error code is returned, exit the program */
 HANDLE_ERROR(status);


/*-------------------------------------------------------------------*/
/* initalize the SCI                                                 */
/*-------------------------------------------------------------------*/

 /* Call SCI_D_Init with valid parameters */
 status = SCI_D_Init(SCIPtr,             /* SCI base address        */
                     19200,              /* use baud rate of 19200  */
                     32000000,           /* SCI Clock is 32 MHz     */
                     SCI_D_PINMODE_CMOS, /* use CMOS pin mode       */
                     SCI_D_IDLE_LONG,    /* use long idle mode      */
                     SCI_D_PARITY_NONE,  /* no parity               */
                     SCI_D_DATA_8,       /* 8 bit data frame        */
                     SCI_D_WAKEUP_IDLE,  /* use idle wakeup mode    */
                     SCI_D_DISABLE       /* disable SCI in doze mode*/
                    );

 /* If an error code is returned, exit the program */
 HANDLE_ERROR(status);

/*-------------------------------------------------------------------*/
/* enable the transmitter and receiver                               */
/*-------------------------------------------------------------------*/

/* Call SCI_D_ControlOperation with valid parameters */
 status = SCI_D_ControlOperation(SCIPtr,       /* SCI base address  */
                                 SCI_D_ENABLE, /* enable transmitter*/
                                 SCI_D_ENABLE  /* enable receiver   */
                                );

/*-------------------------------------------------------------------*/
/* transmit welcome message                                          */
/*-------------------------------------------------------------------*/
```

```
 out_string(SCIPtr, "\r\n\nSCI Echo Program ready...\r\n\n");

/*-------------------------------------------------------------------*/
/* Loop forever, echoing input                                       */
/*-------------------------------------------------------------------*/
 for(;;)
 {

/*-------------------------------------------------------------------*/
/* Wait for a character to be received                               */
/*-------------------------------------------------------------------*/
    while (readyFlag == SCI_D_FALSE)
    {
       /* try to receive */
       status = SCI_D_Receive ( SCIPtr,    /* SCI base address      */
                                receivePtr,/* pointer to return val */
                                statusPtr  /* structure for status  */
                              );
       /* If an error code is returned, exit the program */
       HANDLE_ERROR(status);

       /* check reception status */
       if ((statusPtr->receiveDataPending == SCI_D_TRUE) ||
           (statusPtr->dataOverrun == SCI_D_TRUE) ||
           (statusPtr->noiseDetected == SCI_D_TRUE) ||
           (statusPtr->framingError == SCI_D_TRUE) ||
           (statusPtr->parityError == SCI_D_TRUE) ||
           (statusPtr->receiverActive == SCI_D_TRUE))
       {
         /* Did not successfully receive data, */
         /* keep trying to receive */
       }
       else
       {
         /* successful reception, continue */
         readyFlag = SCI_D_TRUE;
       } /* end if-else */

    } /* end while loop */

    readyFlag = SCI_D_FALSE; /* reset readyFlag */

/*-------------------------------------------------------------------*/
/* End program if a CTRL-D or ESC character was received             */
/*-------------------------------------------------------------------*/
    if ((receiveVal == EOT) || (receiveVal == ESC))
    {
       out_string(SCIPtr, "\n\nGoodbye!\r\n\n");

       /* disable transmitter and receiver */
       /* Call SCI_D_ControlOperation with valid parameters */
       status = SCI_D_ControlOperation(
                              SCIPtr,       /* SCI base address    */
                              SCI_D_DISABLE,/* disable transmitter */
```

```
                                SCI_D_DISABLE /* disable receiver    */
                               );

        /* break out of loop and go to end of function */
        break;
    } /* end if */

 /*-------------------------------------------------------------------*/
 /* Copy character to line echo buffer                                */
 /*-------------------------------------------------------------------*/
    *lineBufferPtr++ = (char)receiveVal;

 /*-------------------------------------------------------------------*/
 /* Echo the character back through the transmitter                   */
 /*-------------------------------------------------------------------*/
    status = SCI_D_Transmit( SCIPtr,
                             receiveVal,
                             transmitDataPendingPtr
                           );

 /*-------------------------------------------------------------------*/
 /* If a carriage return character was received, echo the buffered    */
 /* line and reset line buffer.                                       */
 /*-------------------------------------------------------------------*/
    if (receiveVal == '\r')
    {
        out_string (SCIPtr, "\n");  /* echo line feed */
        if (lineBufferPtr != lineBuffer + 1) /* non-empty line */
        {
            *lineBufferPtr++ = '\n'; /* add line feed to buffer */
            *lineBufferPtr++ = EOS;  /* add end of string char  */
            out_string(SCIPtr, lineBuffer);
        }
        lineBufferPtr = lineBuffer;   /* reset buffer pointer */
    } /* end if */

  } /* end of for loop */
  return (status);

} /* end of main */


/*********************************************************************
   Name: out_string
   Description: Send a string out through the SCI_D transmitter
   Inputs:  SCIptr    (pointer to SCI_D base address)
            str       (character string to transmit)
   Outputs: status (if passed, 0; else line number of the failure)
*********************************************************************/
static void out_string (pSCI_D_t SCIPtr, char * str)
{
 /*-------------------------------------------------------------------*/
 /* declare local variables                                           */
 /*-------------------------------------------------------------------*/
```

```
 /* return codes */
 SCI_D_ReturnCode_t status = SCI_D_ERR_NONE; /* return value        */

 /* for transmission */
 SCI_D_Boolean_t transmitDataPending;      /* status boolean        */
 SCI_D_Boolean_t * transmitDataPendingPtr = &transmitDataPending;

 SCI_D_Boolean_t readyFlag = SCI_D_FALSE;  /* flag for while loop   */

 SCI_D_Status_t statusVal;
 pSCI_D_Status_t statusPtr = &statusVal;   /* for status            */

 char * cp;                                /* string pointer        */

/*-------------------------------------------------------------------*/
/* Loop through string                                               */
/*-------------------------------------------------------------------*/
 for (cp = str; *cp != EOS; cp++)
 {

/*-------------------------------------------------------------------*/
/* Wait for pending transmissions to complete                        */
/*-------------------------------------------------------------------*/
   while (readyFlag == SCI_D_FALSE)
   {
      status = SCI_D_GetStatus(SCIPtr,
                               statusPtr);

      /* check transmission status */
      if (statusPtr->transmitDataPending == SCI_D_TRUE)
      {
          /* can't transmit yet */
      }
      else
      {
          /* continue */
          readyFlag = SCI_D_TRUE;
      } /* end if-else */

  } /* end while loop */

  readyFlag = SCI_D_FALSE; /* reset readyFlag */

/*-------------------------------------------------------------------*/
/* Transmit character                                                */
/*-------------------------------------------------------------------*/
   status = SCI_D_Transmit(SCIPtr,
                           *cp,
                           transmitDataPendingPtr);
 } /* end for loop */

} /* end out_string */
```

Section 17 Watchdog_A Level 1

This section describes the Watchdog_A Level 1 M•CORE™ Device Driver.

17.1 Overview

The Watchdog Timer and its accompanying driver (Watchdog_A) are used on the MMC2107 development system. The level 1 service is designed to allow the programmer to control the Watchdog Timer on a near-register level, but without the concern of exact positions of registers and fields, or the detailed sequences of register operations. Refer to the MMC2107 Advance Information (Data Book) (MMC2107/D) for additional information.

17.2 Watchdog_A Service Functions

The Watchdog_A device driver software provides these functions:

- Initializes the Watchdog Timer.
- Determines the timer modulus that is reloaded into the counter after proper service sequence.
- Gets the contents of the Watchdog Timer count register.
- Controls the restarting of the watchdog counter to keep it from timing out and causing a reset.
- Writes unformatted data to selected writeable Watchdog Timer registers.
- Gets unformatted data from specified readable Watchdog Timer registers.

17.3 Watchdog_A Data Type Definitions

The following paragraphs describe the data definitions recognized by the Watchdog_A device driver. These data definitions are grouped into six categories: standard data types, board specific base address definitions, derived data types, enumerated data types, register macros, and return codes.

17.3.1 Standard Data Type Definitions

- UINT8 — 8-bit unsigned data value
- INT8 — 8-bit signed data value
- UINT16 — 16-bit unsigned data value
- INT16 — 16-bit signed data value
- UINT32 — 32-bit unsigned data value
- INT32 — 32-bit signed data value

17.3.2 Board Specific Base Address Definitions

```
#define __MMC2107_WATCHDOG   (0x00C70000)
```

17.3.3 Derived Data Type Definitions

17.3.3.1 Watchdog_A_t - Watchdog Timer Register Definition

```
typedef struct {
            UINT16    WCR;      /* Watchdog control register */
   volatile UINT16    WMR;      /* Watchdog modulus register */
            UINT16    WCNTR;    /* Watchdog counter register */
            UINT16    WSR;      /* Watchdog service register */
} Watchdog_A_t, *pWatchdog_A_t;
```

17.3.4 Enumerated Data Type Definitions

17.3.4.1 Watchdog_A_Registers_t - Watchdog Timer Register Selection

```
typedef enum {
    WATCHDOG_A_WCR,      /* Select WATCHDOG control Register  */
    WATCHDOG_A_WMR,      /* Select WATCHDOG modulus Register  */
    WATCHDOG_A_WCNTR,    /* Select WATCHDOG counter Register  */
    WATCHDOG_A_WSR       /* Select Service Register           */
} Watchdog_A_Registers_t;
```

17.3.4.2 Watchdog_A_Control_t – Watchdog enable selection

```
typedef enum {
    WATCHDOG_A_DISABLE,   /* Disable Watchdog Timer */
    WATCHDOG_A_ENABLE     /* Enable Watchdog Timer  */
} Watchdog_A_Control_t;
```

17.3.4.3 Watchdog_A_Debug_t - Watchdog Debug mode control

```
typedef enum {
 WATCHDOG_A_FUNCTION_UNAFFECTED_DEBUG_MODE, /* Function not affected  */
                                            /* during DEBUG mode      */
 WATCHDOG_A_FUNCTION_STOPPED_DEBUG_MODE /* Function stopped during DEBUG mode */
} Watchdog_A_Debug_t;
```

17.3.4.4 Watchdog_A_DOZE_t – DOZE Mode Selection.

```
typedef enum {
   WATCHDOG_A_FUNCTION_UNAFFECTED_DOZE_MODE,  /* Function not affected  */
                                              /* during DOZE mode       */
   WATCHDOG_A_FUNCTION_STOPPED_DOZE_MODE   /* Function stopped during DOZE mode */
} Watchdog_A_DOZE_t;
```

17.3.4.5 Watchdog_A_Wait_t – Wait Mode Selection.

```
typedef enum {
   WATCHDOG_A_FUNCTION_UNAFFECTED_WAIT_MODE,    /* Function not affected  */
                                                /* during Wait mode       */
   WATCHDOG_A_FUNCTION_STOPPED_WAIT_MODE  /* Function stopped during Wait mode */
} Watchdog_A_Wait_t;
```

17.3.5 Register Macros

17.3.5.1 Watchdog Control Register

```
#define WCR_EN_BITNO 0
#define WCR_EN_MASK (1  <<  WCR_EN_BITNO)
#define WCR_DBG_BITNO 1
#define WCR_DBG_MASK (1  <<  WCR_DBG_BITNO)
#define WCR_DOZE_BITNO 2
#define WCR_DOZE_MASK (1  <<  WCR_DOZE_BITNO)
#define WCR_WAIT_BITNO 3
#define WCR_WAIT_MASK (1  <<  WCR_WAIT_BITNO)

#define WCR_RESET_MASK 0x000F
```

17.3.5.2 Watchdog Modulus Register

```
#define WMR_WM_MAX 0xFFFF
#define WMR_WM_BITNO 0
#define WMR_WM_MASK (WMR_WM_MAX << WMR_WM_BITNO)

#define WMR_RESET_MASK 0xFFFF
```

17.3.5.3 Watchdog Count Register

```
#define WCNTR_WC_MAX 0xFFFF
#define WCNTR_WC_BITNO 0
#define WCNTR_WC_MASK (WCNTR_WC_MAX << WCNTR_WC_BITNO)
```

17.3.5.4 Watchdog Service Register

```
#define WSR_WS_MAX 0xFFFF
#define WSR_WS_BITNO 0
#define WSR_WS_MASK (WSR_WS_MAX << WSR_WS_BITNO)

#define WSR_RESET_MASK 0x0000
#define WSR_FIRST_RESTART_VALUE  0x5555
#define WSR_SECOND_RESTART_VALUE 0xAAAA
```

17.3.6 Return Codes

Return codes from the enumerated type “Watchdog_A_ReturnCode_t”.

Return Code	Functions Returning	Description
WATCHDOG_A_ERR_NONE	All	No error.
WATCHDOG_A_ERR_INVALID_HANDLE	All	WATCHDOG base address parameter is zero.
WATCHDOG_A_ERR_BAD_RESULT_ADDR	GetRegister, GetCounterValue	Result Pointer is zero.
WATCHDOG_A_ERR_INVALID_REGISTER	SetRegister, GetRegister	WATCHDOG Register Selection is invalid.
WATCHDOG_A_ERR_INVALID_ENABLE_VALUE	Watchdog_A_Init	The enable mode value is out of range.
WATCHDOG_A_ERR_INVALID_DEBUG_MODE_VALUE	Watchdog_A_Init	The DEBUG mode value is out of range.
WATCHDOG_A_ERR_INVALID_DOZE_MODE_VALUE	Watchdog_A_Init	The DOZE mode value is out of range.
WATCHDOG_A_ERR_INVALID_WAIT_MODE_VALUE	Watchdog_A_Init	The Wait mode value is out of range.

17.4 Watchdog_A API Definitions

17.4.1 Watchdog_A_Init

17.4.1.1 Description

The Watchdog_A_Init function initializes the Watchdog Timer. Parameter checking is a compile-time option.

This function should be called during the system software initialization or for reinitialization of the module. The function can only be called with supervisor accessibility.

17.4.1.2 Prototype

```
Watchdog_A_ReturnCode_t  Watchdog_A_Init (
          pWatchdog_A_t           WATCHDOGPtr,
          Watchdog_A_Control_t    EnableValue,
          Watchdog_A_Debug_t      DebugModeValue,
          Watchdog_A_DOZE_t       DOZEModeValue,
          Watchdog_A_Wait_t       WaitModeValue
)
```

17.4.1.3 Arguments

Data Type	Formal Name	Description	Valid Values
pWatchdog_A_t	WATCHDOGPtr	Watchdog Timer base address associated with this driver.	Any non-zero core processor data address.
Watchdog_A_Control_t	EnableValue	Enable/Disable Watchdog Timer.	WATCHDOG_A_DISABLE, WATCHDOG_A_ENABLE
Watchdog_A_Debug_t	DebugModeValue	Control function of the Watchdog Timer in DEBUG mode.	WATCHDOG_A_FUNCTION_UNAFFECTED_DEBUG_MODE, WATCHDOG_A_FUNCTION_STOPPED_DEBUG_MODE
Watchdog_A_DOZE_t	DOZEModeValue	Control function of the Watchdog Timer in DOZE mode.	WATCHDOG_A_FUNCTION_UNAFFECTED_DOZE_MODE, WATCHDOG_A_FUNCTION_STOPPED_DOZE_MODE
Watchdog_A_Wait_t	WaitModeValue	Control function of timer in wait mode.	WATCHDOG_A_FUNCTION_UNAFFECTED_WAIT_MODE, WATCHDOG_A_FUNCTION_STOPPED_WAIT_MODE

17.4.1.4 Return Values

Error Code	Description
WATCHDOG_A_ERR_NONE	No error
WATCHDOG_A_ERR_INVALID_HANDLE	Watchdog Timer base address parameter is zero.
WATCHDOG_A_ERR_INVALID_ENABLE_VALUE	The enable mode value is out of range.
WATCHDOG_A_ERR_INVALID_DEBUG_MODE_VALUE	The DEBUG mode value is out of range.
WATCHDOG_A_ERR_INVALID_DOZE_MODE_VALUE	The DOZE mode value is out of range.
WATCHDOG_A_ERR_INVALID_WAIT_MODE_VALUE	The Wait mode value is out of range.

17.4.2 Watchdog_A_ConfigureModulus

17.4.2.1 Description

The Watchdog_A_ConfigureModulus function determines the timer modulus that is reloaded into the counter after proper service sequence. Parameter checking is a compile-time option.

This function should only be called after the Watchdog Timer has been initialized through a previous call to the Watchdog_A_Init function. TheWatchdog_A_ConfigureModulus function can only be called with supervisor accessibility.

17.4.2.2 Prototype

```
Watchdog_A_ReturnCode_t Watchdog_A_ConfigureModulus (
          pWatchdog_A_t    WATCHDOGPtr,
          UINT16           ModulusValue
)
```

17.4.2.3 Arguments

Data Type	Formal Name	Description	Valid Values
pWatchdog_A_t	WATCHDOGPtr	WATCHDOG base address associated with this driver.	Any non-zero core processor data address.
UINT16	ModulusValue	Location in which to load counter value.	Any non-zero core processor data address.

17.4.2.4 Return Values

Error Code	Description
WATCHDOG_A_ERR_NONE	No error
WATCHDOG_A_ERR_INVALID_HANDLE	Watchdog Timer base address parameter is zero

17.4.3 Watchdog_A_GetCounterValue

17.4.3.1 Description

The Watchdog_A_GetCounterValue function returns the contents of the Watchdog Timer count register. Parameter checking is a compile-time option.

This function can be called at any time.

17.4.3.2 Prototype

```
Watchdog_A_ReturnCode_t Watchdog_A_GetCounterValue (
          pWatchdog_A_t     WATCHDOGPtr,
          UINT16           *CounterValuePtr
)
```

17.4.3.3 Arguments

Data Type	Formal Name	Description	Valid Values
pWatchdog_A_t	WATCHDOGPtr	WATCHDOG base address associated with this driver.	Any non-zero core processor data address.
UINT16 *	CounterValuePtr	Location in which to store counter value.	Any non-zero 16-bit value

17.4.3.4 Return Values

Error Code	Description
WATCHDOG_A_ERR_NONE	No error
WATCHDOG_A_ERR_INVALID_HANDLE	WATCHDOG base address parameter is zero
WATCHDOG_A_ERR_BAD_RESULT_ADDR	Result Pointer is zero

17.4.4 Watchdog_A_ServiceWatchdog

17.4.4.1 Description

The Watchdog_A_ServiceWatchdog function controls the restarting of the watchdog counter to keep it from timing out and causing a reset. Parameter checking is a compile-time option.

17.4.4.2 Prototype

```
Watchdog_A_ReturnCode_t Watchdog_A_ServiceWatchdog (
          pWatchdog_A_t     WATCHDOGPtr
)
```

17.4.4.3 Arguments

Data Type	Formal Name	Description	Valid Values
pWatchdog_A_t	WATCHDOGPtr	WATCHDOG base address associated with this driver.	Any non-zero core processor data address.

17.4.4.4 Return Values

Error Code	Description
WATCHDOG_A_ERR_NONE	No error
WATCHDOG_A_ERR_INVALID_HANDLE	Watchdog base address parameter is zero

17.4.5 Watchdog_A_SetRegister

17.4.5.1 Description

The Watchdog_A_SetRegister function writes unformatted data to specified writeable Watchdog Timer registers. Parameter checking is a compile-time option.

This function can be called at any time.

17.4.5.2 Prototype

```
Watchdog_A_ReturnCode_t   Watchdog_A_SetRegister (
          pWatchdog_A_t             WATCHDOGPtr,
          Watchdog_A_Registers_t    WatchdogRegister,
          UINT16                    RegisterValue
)
```

17.4.5.3 Arguments

Data Type	Formal Name	Description	Valid Values
pWatchdog_A_t	WATCHDOGPtr	Watchdog_A base address associated with this driver.	Any non-zero core processor data address.
Watchdog_A_Registers_t	WatchdogRegister	Select among Watchdog_A registers.	WATCHDOG_A_WCR WATCHDOG_A_WMR WATCHDOG_A_WSR
UINT16	RegisterValue	Data to copy into selected writeable Watchdog_A register.	Any 16-bit value.

17.4.5.4 Return Values

Error Code	Description
WATCHDOG_A_ERR_NONE	No error
WATCHDOG_A_ERR_INVALID_HANDLE	WATCHDOG base address parameter is zero
WATCHDOG_A_ERR_INVALID_REGISTER	WATCHDOG Register Selection is invalid

17.4.6 Watchdog_A_GetRegister

17.4.6.1 Description

The Watchdog_A_GetRegister function returns unformatted data from selected readable Watchdog Timer registers. Parameter checking is a compile-time option.

This function can be called at any time.

17.4.6.2 Prototype

```
Watchdog_A_ReturnCode_t   Watchdog_A_GetRegister (
          pWatchdog_A_t               WATCHDOGPtr,
          Watchdog_A_Registers_t    WatchdogRegister,
          UINT16                   *GetRegisterPtr
)
```

17.4.6.3 Arguments

Data Type	Formal Name	Description	Valid Values
pWatchdog_A_t	WATCHDOGPtr	Watchdog_A base address associated with this driver.	Any non-zero core processor data address.
Watchdog_A_Registers_t	WatchdogRegister	Select among a sub-set of Watchdog_A registers.	WATCHDOG_A_WCR WATCHDOG_A_WMR WATCHDOG_A_WCNTR WATCHDOG_A_WSR
UINT16 *	GetRegisterPtr	Result address for selected Watchdog_A register data.	Any non-zero core processor data address.

17.4.6.4 Return Values

Error Code	Description
WATCHDOG_A_ERR_NONE	No error
WATCHDOG_A_ERR_INVALID_HANDLE	WATCHDOG base address parameter is zero
WATCHDOG_A_ERR_BAD_RESULT_ADDR	Result Pointer is zero
WATCHDOG_A_ERR_INVALID_REGISTER	WATCHDOG Register Selection is invalid

17.5 Watchdog_A Example Application

The following describes the Watchdog_A example application.

17.5.1 Description for Example Application

This program illustrates use of the Watchdog Timer and Watchdog_A driver. The example application first initializes the Watchdog Timer, then demonstrates how to configure the watchdog, initialize the modulus, obtain the counter value, service the watchdog, and obtain the counter value again.

17.5.1.1 Example Application Code

```
/*********************************************************************

File:    watchdoga_app.c

Purpose: The following example code illustrates the use of the M-CORE
         Peripherals Library Watchdog_A Level 1 driver.

         This program illustrates use of the Watchdog Timer. It will
         first initialize the watchdog timer, demonstrate how to
         configure the watchdog, initialize the modulus, and obtain
         the counter value, service the watchdog, and obtain the
         counter value again.

         If an error occurs, the program will return with the line
         number of where the failure occurred.

Time to run:
         Less than 30 seconds.

General Setup:
         General hardware and software setup to run this application
         can be found in the 'readme_app.txt' file, located in the
         '<library>/app' directory.

Application Specific Setup:
         Development Board: MMC2107 CMB or EVB

(C) Copyright Motorola Inc, 2000.  All rights reserved.
*********************************************************************/

#include "watchdog_a.h"
#include "plibdefs_mmc2107.h"

/* macro to handle error checking and response */
#define HANDLE_ERROR(err)   if (err != WATCHDOG_A_ERR_NONE) return(__LINE__);
#define ModulusValue 0xBEAF

int main(void);                /* function prototype */
int watchdog_example(pWatchdog_A_t);     /* function prototype */
```

```
/*********************************************************************
   Name: main
   Description: Drive program operation
   Inputs: WATCHDOGPtr (pointer to WATCHDOG_A base address)
   Outputs: int status
*********************************************************************/
int main ()
{
    int status;

    status = watchdog_example ((pWatchdog_A_t)__MMC2107_WATCHDOG);

    return (status);
}

/*********************************************************************
   Name: watchdog_example
   Description: Provide an example for the usage of the Watchdog_A
   Inputs:  WATCHDOGPtr    (pointer to Watchdog_A base address)
   Outputs: status (if passed, 0; else line number of the failure)
*********************************************************************/
int watchdog_example (pWatchdog_A_t WATCHDOGPtr)
{

  Watchdog_A_ReturnCode_t   retval;      /* return value           */

UINT16 CounterValue;
UINT16 * CounterValuePtr = &CounterValue;


  /* Call Watchdog_A_ConfigureModulus with valid parameters */
  retval = Watchdog_A_ConfigureModulus  (
                   WATCHDOGPtr,
                   ModulusValue );

  /* If an error code is returned, exit the program */
  HANDLE_ERROR(retval);

  /* Call Watchdog_A_ServiceWatchdog with valid parameters */
  retval = Watchdog_A_ServiceWatchdog (
                                       WATCHDOGPtr  );

  /* If an error code is returned, exit the program */
  HANDLE_ERROR(retval);

  /* Call Watchdog_A_GetCounterValue with valid parameters */
  retval = Watchdog_A_GetCounterValue   (
                                          WATCHDOGPtr,
                                          CounterValuePtr
                                        );

  HANDLE_ERROR(retval);
```

```
  if (CounterValue > ModulusValue)
      {
       return (__LINE__);
      } /* end if */

  return(retval);

} /* end of main */
```

Appendix A Definitions and Acronyms

Table A-1 Definitions And Acronyms

Term/Acronym	Definition
API	**A**pplications **P**rogram **I**nterface
AMD	**A**dvanced **M**icro **D**evices
CMFR	**C**DR **M**oneT **F**lash EEPROM for **R**LB Module
CMB	micro-**C**ontroller **M**emory **B**oard. The emulation processor plugs in here.
CCM	**C**hip **C**onfiguration **M**odule
COM	**COM**munications Port
CPU	**C**entral **P**rocessing **U**nit
CS	**C**hip **S**elects
EBDI	**E**nhanced **B**ackground **D**ebug **I**nterface
EVB	**Ev**aluation **B**oard
EVS	**Ev**aluation **S**ystem
FLASH	Reprogrammable Nonvolatile Memory
FSRAM	**F**ast **S**tatic **R**andom **A**ccess **M**emory
GNU	A suite of free software development tools provided by Motorola.
GPIO	**G**eneral **P**urpose **I/O**
I/O	**I**nput **O**utput
ITCN	**I**nterrupt **C**ontroller
Level 1 Service	A service that provides basic register-level access to a peripheral module, providing minimal interrupt support. Maintains no state information. The abstract device provided is almost identical to the actual hardware.
Level 2 Service	A service that provides a higher level of abstraction than a level 1 service, typically including interrupt service (where appropriate) to provide that abstraction. Example: a queued character I/O system built upon an SCI device.
MBISM	**M**IOS **B**us **I**nterface **S**ub**M**odule
M•CORE M-CORE	32-bit microRISC processor designed for high performance with low energy consumption (formerly known as the RISC Communication Engine or RCE).
MCU	**M**icro**C**ontroller **U**nit. A silicon device containing a CPU, memory, timers, and a number of peripheral devices that are used in embedded system applications.
MDASM	**M**IOS **D**ual **A**ction **S**ub**M**odule
MIOS	**M**odular **I/O S**ubsystem
MPFB	**M**odular **P**lat**F**orm **B**oard
OnCE	**On C**hip **E**mulation
PC	**P**ersonal **C**omputer
PIT	**P**rogrammable **I**nterval **T**imer
PLL	**P**hase **L**ock **L**oop
RAM	**R**andom **A**ccess **M**emory
ROM	**R**ead **O**nly **M**emory
QADC64	**Q**ueued **A**nalog to **D**igital **C**onverter (**64** entries**)**

Appendix A

Table A-1 Definitions And Acronyms

Term/Acronym	Definition
SCI	**S**erial **C**ommunication **I**nterface **M**odule
SPI	**S**erial **P**eripheral **I**nterface
SRAM	**S**tatic **R**andom **A**ccess **M**emory

D

E

S

V

W

Revision History

The MMC2107 Device Driver Library Reference Manual is a new document.