PIC32 Peripheral Libraries for
MPLAB C32 Compiler
# TABLE OF CONTENTS

1.0 Introduction.......................................................................................................................... 1
2.0 System Functions.................................................................................................................. 3
2.0 Pcache Functions.................................................................................................................. 7
3.0 DMA Functions................................................................................................................... 17
4.0 Bus Matrix Functions.......................................................................................................... 53
5.0 NVM Functions................................................................................................................... 57
6.0 Reset Functions................................................................................................................... 59
7.0 Interrupt Functions.............................................................................................................. 69
8.0 Oscillator Functions........................................................................................................... 91
9.0 Power Save Functions......................................................................................................... 97
10.0 I/O Port library.................................................................................................................. 99
11.0 Timer Functions.............................................................................................................. 153
12.0 Input Capture Functions.................................................................................................. 175
13.0 Output Compare Functions............................................................................................... 183
14.0 SPI Functions.................................................................................................................. 193
15.0 I2C™ Functions............................................................................................................. 215
16.0 UART Functions.............................................................................................................. 231
17.0 PMP Functions.............................................................................................................. 243
19.1 RTCC Functions............................................................................................................. 263
18.0 ADC10 Functions........................................................................................................... 291
19.0 Comparator Functions...................................................................................................... 301
20.0 CVREF Functions........................................................................................................... 305
21.0 WDT Functions.............................................................................................................. 309
Chapter 1. 32-Bit Peripheral Libraries

1.0 INTRODUCTION

This chapter documents the functions and macros contained in the 32-bit peripheral libraries. Examples of use are also provided.

1.1 C Code Applications

The MPLAB C32 C compiler install directory (c:\Program Files\Microchip\MPLAB C32) contains the following subdirectories with library-related files:

- pic32mx\include\plib.h - Master include file for all APIs
- pic32mx\include\peripheral\*.h - API header files
- pic32-libs\peripheral\*.c - library source files

1.2 Chapter Organization

This chapter is organized as follows:

- Using the 32-Bit Peripheral Libraries
- Individual Peripheral Module functions and macros
  - System Level Functions
  - Prefetch Cache Functions
  - DMA Functions
  - Bus Matrix Functions
  - NVM Functions
  - Reset/Control Functions
  - Interrupt Functions
  - Oscillator Functions
  - Power Save Functions
  - I/O Port Functions
  - Timer Functions
  - Input Capture Functions
  - Output Compare Functions
  - SPI Functions
  - I2C™ Functions
  - UART Functions
  - PMP Functions
  - RTCC Functions
  - A/D Functions
  - Comparator Functions
  - CVREF Functions
  - Watchdog Timer Functions
1.3 Using the 32-Bit Peripheral Libraries

Applications wishing to use peripheral libraries need to include <plib.h> file in their source file. The C32 compiler has built-in knowledge of all header file and library files.

The master header file plib.h, includes all individual peripheral header files. An application needs to include plib.h only to access any of the supported functions and macros. If you need to refer to individual header file content, they are located in pic32mx\include\peripheral folder in your C32 installation directory. Complete source is located in pic32-libs\peripheral folder.

If required, you may rebuild the peripheral libraries. Please follow the procedure outlined in the text file located in the pic32-libs directory.
2.0 SYSTEM FUNCTIONS

The PIC32MX system library consists of functions and macros to perform system level operations.

SYSTEMConfigPerformance
SYSTEMConfigWaitStatesAndPB
SYSTEMConfigPB
## 2.1 SYSTEM Functions

### SYSTEMConfigPerformance

**Description:** This function automatically configures the device for maximum performance for a given system clock frequency

**Include:** plib.h

**Prototype:**

```c
void SYSTEMConfigPerformance(unsigned int sys_clock);
```

**Arguments:**

- `sys_clock` The system clock in Hz

**Return Value:** None

**Remarks:** This function configures the Flash Wait States, Data Wait States and PBCLK divider to lowest value allowed for the given system clock. If the device has Prefetch-Cache module, it also enables prefetch and cache mode. In summary, this function configures all necessary parameters to achieve maximum performance for given system clock

**Code Example:**

```c
SYSTEMConfigPerformance(72000000);
```

### SYSTEMConfigWaitStatesAndPB

**Description:** This function automatically configures flash wait states and PBCLK divider for a given system frequency.

**Include:** plib.h

**Prototype:**

```c
void SYSTEMConfigWaitStatesAndPB(unsigned int sys_clock);
```

**Arguments:**

- `sys_clock` The system clock in Hz

**Return Value:** None

**Remarks:** This function configures flash wait states and PBCLK divider to lowest value allowed for the given system clock. It does not configure prefetch cache module.

**Source File:**

**Code Example:**

```c
SYSTEMConfigWaitStatesAndPB(72000000);
```

### SYSTEMConfigPB

**Description:** This function automatically configures PBCLK divider for a given system frequency.

**Include:** plib.h

**Prototype:**

```c
void SYSTEMConfigPB(int sys_clock);
```

**Arguments:**

- `sys_clock` The system clock in Hz

**Return Value:** None
<table>
<thead>
<tr>
<th>Remark</th>
<th>SYSTEMConfigPB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Remarks:</strong></td>
<td>This function configures the PBCLK divider to lowest value allowed for the given system clock.</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Code Example:</strong></td>
<td>SYSTEMConfigPB(72000000);</td>
</tr>
</tbody>
</table>
2.2 Example: Using SYSTEMConfigPerformance

The following code example illustrates how to configure the device for maximum performance for a given system clock.

```c
#include <plib.h>

// Configuration Bit settings
// SYSCLK = 72 MHz (8MHz Crystal/ FPLLIDIV * FPLLMUL / FPLLODIV)
// PBCLK = 36 MHz
// Primary Osc w/PLL (XT+, HS+, EC+PLL)
// WDT OFF
// Other options are don't care

#pragma config FPLLMUL = MUL_18, FPLLIDIV = DIV_2
#pragma FPLLODIV = DIV_1, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRIPLL, FPBDIV = DIV_2

int main(void)
{
    /*
    Configure the device for maximum performance.
    This macro sets flash wait states, PBCLK divider and DRM wait states based on the specified lock frequency. It also turns on the cache mode if available.
    Based on the current frequency, the PBCLK divider will be set at 1:2. This knowledge is required to correctly set UART baud rate, timer reload value and other time sensitive setting.
    */
    SYSTEMConfigPerformance(72000000L);

    // Use PBCLK divider of 1:2 to calculate UART baud, timer tick etc.
    ...
}
```
2.0 PCACHE FUNCTIONS

The PIC32MX Pcache library consists of functions and macros supporting common configuration and control features of this peripheral set.

PrefetchCache Operations
cheConfigure
mCheConfigure
mCheGetCon
mCheSetCacheAccessLine
mCheGetAcc
mCheSetCacheTag
mCheGetCacheTag
mCheSetMask
mCheGetMask
mCheWriteCacheLine
mCheInvalidateLine
mCheInvalidateAllLines
mCheLockLine
mCheGetHit
mCheGetMis
CheKseg0CacheOff
CheKseg0CacheOn
2.1 Prefetch Cache Functions and Macros

cheConfigure

**Description:**
This macro is identical to mCheConfigure except that it accepts individual parameters instead of a bit-mask of parameters.

**Include:**
plib.h

**Prototype:**
```c
void cheConfigure(int checoh, int dcsz, int prefen, int pfmws);
```

**Arguments:**
- `checoh` Cache coherency (1 = Coherent, 0 = Incoherent)
- `dcsz` Data cache line size (a value between 0 - x)
- `prefen` Prefetch enable (1 = enable, 0 = disable)
- `pfmws` Flash Memory wait states (0 - 7)

**Return Value:**
None

**Remarks:**
This function accepts individual prefetch configuration values and initializes the prefetch modules accordingly.

**Code Example:**
```c
// Invalidate cache, 2 data cache lines, prefetch enable, Flash memory wait states of 2
cheConfigure(0, 2, 1, 2);
```

mCheConfigure

**Description:**
This macro provides a second method to configure the prefetch cache module

**Include:**
plib.h

**Prototype:**
```c
void mCheConfigure(config);
```

**Arguments:**
- `config` This argument contains one or more bit masks bitwise OR'd together. Select one or more from the following bit masks. Note: An absent mask symbol in an argument assumes the corresponding bit(s) are disabled, or default value, and are set = 0.

- **Coherency during Flash Programming**
  - CHE_CONF_COH_INVUNL
  - CHE_CONF_COH_INVALL
  (These bit fields are mutually exclusive)

- **Data Cache Lines**
  - CHE_CONF_DC_NONE
  - CHE_CONF_DC_1LINE
  - CHE_CONF_DC_2LINES
  - CHE_CONF_DC_4LINES
  (These bit fields are mutually exclusive)

- **Prefetch Behavior**
  - CHE_CONF_PF_DISABLE
  - CHE_CONF_PF_C
  - CHE_CONF_PF_NC
  - CHE_CONF_PF_ALL
  (These bit fields are mutually exclusive)
mCheConfigure (Continued)

Flash Wait States
CHE_CONF_WS0
CHE_CONF_WS1
CHE_CONF_WS2
CHE_CONF_WS3
CHE_CONF_WS4
CHE_CONF_WS5
CHE_CONF_WS6
CHE_CONF_WS7
(These bit fields are mutually exclusive)

Return Value: None
Remarks: This function loads the checon register with concatenation of the arguments.

Code Example: mCheConfigure(CHE_CONF_PF_C | CHE_CONF_WS2);

mCheGetCon

Description: This macro returns the current value of the CHECON register
Include: plib.h
Prototype: void mCheGetCon(void);

Arguments: None
Return Value: The 32-bit value of the CHECON register
Remarks:
Code Example: cur_wait_states = mCheGetCon() & 0x7;

mCheSetCacheAccessLine

Description: This macro is used to set up the CHEACC register. The value of the CHEACC register is used as an index during any access to cache line information such as tags, masks, or data words.
Include: plib.h
Prototype: void mCheSetCacheAccessLine(int idx, int writeEnable);

Arguments: idx - Index of the cache line to access (0-15)
writeEnable - ‘1’ Enables writes to the cache line (tags, mask, and data words), ‘0’ disables it

Return Value: None
Remarks: This macro is invoked implicitly by using many of the other macros in this package
Code Example: mCheSetCacheAccessLine(12,1);
mCheGetAcc

**Description:** This macro returns the current value of the CHEACC register

**Include:** plib.h

**Prototype:**

```c
void mCheGetAcc(void);
```

**Arguments:** None

**Return Value:** The 32-bit value of the CHEACC register

**Remarks:**

**Code Example:**

```c
curidx = mCheGetAcc() & 0xf;
```

mCheSetCacheTag

**Description:** This macro writes a tag entry into a single line of the prefetch cache.

**Include:** plib.h

**Prototype:**

```c
void mCheSetCacheTag(int lineno, unsigned addr, unsigned attr);
```

**Arguments:**

- **lineno** Index of the cache line to access (0-15)
- **addr** Physical address that corresponds to this cache line
- **attr** This argument contains one or more bit masks bitwise OR’d together. Select one or more from the following bit masks. Note: An absent mask symbol in an argument assumes the corresponding bit(s) are disabled, or default value, and are set = 0

  - **Line Valid**
    - CHE_TAG_INVALID
    - CHE_TAG_VALID
  
  (These bit fields are mutually exclusive)

  - **Line Locked**
    - CHE_TAG_UNLOCKED
    - CHE_TAG_LOCKED
  
  (These bit fields are mutually exclusive)

  - **Line Type**
    - CHE_TAG_TYPE_DATA
    - CHE_TAG_TYPE_INST
  
  (These bit fields are mutually exclusive)

**Return Value:** None

**Remarks:**

The macro sets the tag bits of a single cache line. The cache line corresponding to the lineno parameter is selected automatically by a call to the mCheSetCacheAccessLine macro.

This function must be used carefully. Setting a tag to CHE_TAG_VALID without calling mCheWriteCacheLine() can cause unpredictable results.

**Code Example:**

```c
mCheSetCacheTag(12, 0x1d002f00, CHE_TAG_INVALID | CHE_TAG_LOCKED);
```
mCheGetCacheTag

Description: This macro returns the current value of the CHETAG register

Include: plib.h

Prototype: void mCheGetCacheTag(int lineno);

Arguments: lineno - Index of the cache line to access (0-15)

Return Value: The 32-bit value of CHETAG

Remarks: This macro uses the mCheSetCacheAccessLine macro to select the cache line corresponding to the lineno parameter and then returns the value of CHETAG.

Code Example: tag0 = mCheGetCacheTag(0);

mCheSetMask

Description: This macro writes a mask entry into a single line of the prefetch cache.

Include: plib.h

Prototype: void mCheSetMask(int idx, unsigned mask);

Arguments: idx - Index of the cache line to access (0-15)

mask - this value is written directly to the CHEMSK register of the selected cache line.

Return Value: None

Remarks: The macro sets the mask bits of a single cache line. The cache line corresponding to the idx parameter is selected automatically by a call to the mCheSetCacheAccessLine macro.

This function must be used carefully. Setting a mask value to non-zero causes tag bits to be ignored during cache lookup operations whenever instruction fetches or data reads from the program flash memory occur.

Note: Only cache lines 10 and 11 have CHEMSK registers.

Code Example: mCheSetMask(10, 0x40);
mCheGetMask

Description: This macro returns the current value of the CHEMSK register

Include: plib.h

Prototype: void mCheGetMask(int idx);

Arguments: idx - Index of the cache line to access (0-15)

Return Value: The 32-bit value of CHEMSK

Remarks: This macro uses the mCheSetCacheAccessLine macro to select the cache line corresponding to the lineno parameter and then returns the value of CHEMSK.

Only lines 10 and 11 have writeable CHEMSK registers. All other CHEMSK registers return 0.

Code Example: curmask10 = mCheGetMask(10);

mCheWriteCacheLine

Description: This macro is used to write 4 words of data or instructions to a cache line.

Include: plib.h

Prototype: void mCheWriteCacheLine(unsigned long values[4]);

Arguments: values - the 4 unsigned long values to be written to the selected cache line.

Return Value: None

Remarks: Unlike most of the other functions that write to a cache line, this macro does not automatically select a cache line by calling mCheSetCacheAccessLine(). mCheSetCacheAccessLine() must be called before using this macro.

Code Example: mCheSetCacheAccessLine(12,1);
mCheWriteCacheLine(val_array);
**mCheInvalidateLine**

**Description:** This macro invalidates a single cache line.

**Include:** plib.h

**Prototype:**

```c
void mCheInvalidateLine(int idx);
```

**Arguments:**

- `idx` - Index of the cache line to access (0-15)

**Return Value:** None

**Remarks:**
The macro clears the valid bit in the tag of a single cache line. The cache line corresponding to the `idx` parameter is selected automatically by a call to the `mCheSetCacheAccessLine` macro.

**Code Example:**

```c
mCheInvalidateLine(5);
```

**mCheInvalidateAllLines**

**Description:** This macro invalidates all the lines located in the prefetch cache

**Include:** plib.h

**Prototype:**

```c
void mCheInvalidateAllLines(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:**

**Code Example:**

```c
mCheInvalidateAllLines();
```

**mCheLockLine**

**Description:** This macro causes an automatic fetch and lock of a single cache line.

**Include:** plib.h

**Prototype:**

```c
void mCheLockLine(int idx, int type, unsigned addr);
```

**Arguments:**

- `idx` - Index of the cache line to lock (0-15)
- `type` -
  - 1 - Locks a data line
  - 0 - Locks an instruction line
- `addr` - Physical address that corresponds to this cache line

**Return Value:** None

**Remarks:**
The macro clears the valid bit and sets the lock bit in the tag of a single cache line. The cache line corresponding to the `idx` parameter is selected automatically by a call to the `mCheSetCacheTag` macro.

A cache line marked as locked and not valid will cause the data at the corresponding address to be fetched and locked in the prefetch cache.

**Code Example:**

```c
mCheLockLine(3, 1, 0x1d0030a0);
```
### mCheGetHit

**Description:** This macro returns the current value of the CHEHIT register.

**Include:** `plib.h`

**Prototype:**
```c
void mCheGetHit(void);
```

**Arguments:** None

**Return Value:** The 32-bit value of the CHECON register

**Remarks:**

**Code Example:**
```c
mCheGetCon();
```

### mCheGetMis

**Description:** This macro returns the current value of the CHEMIS register.

**Include:** `plib.h`

**Prototype:**
```c
void mCheGetCon(void);
```

**Arguments:** None

**Return Value:** The 32-bit value of the CHECON register

**Remarks:**

**Code Example:**
```c
mCheGetCon();
```

### CheKseg0CacheOff

**Description:** This function disables cacheing of KSEG0 Program Flash Memory accesses.

**Include:** `plib.h`

**Prototype:**
```c
void CheKseg0CacheOff(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** This function writes the value 2 to the CCA bits in the Config register thereby disabling the cache for code executing within the KSEG0 memory region.

**Code Example:**
```c
CheKseg0CacheOff();
```
### CheKseg0CacheOn

**Description:** This function enables caching of KSEG0 Program Flash Memory accesses.

**Include:** plib.h

**Prototype:**

```c
void CheKseg0CacheOn(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** This function writes the value 3 to the CCA bits in the Config register thereby enabling the cache for code executing within the KSEG0 memory region.

**Source File:**

**Code Example:**

```c
CheKseg0CacheOn();
```
2.2 Prefetch Cache Example

#include <plib.h>

int main(void)
{
    // Set Periph Bus Divider 72MHz / 2 = 36MHz Fpb
    mOSCSetPBDIV(OSC_PB_DIV_2);

    // enable cacheability for KSEG0
    CheKseg0CacheOn();

    // configure the cache for prefetch and 2 wait-state operation
    mCheConfigure(CHE_CONF_WS2 | CHE_CONF_PF_C);

    // The prefetch cache is now configured and ready for use
    ... 

    return 0;
}
3.0 DMA FUNCTIONS

This section provides a list and a description of the interface functions that are part of the DMA API Peripheral Library.

3.1 High level DMA channel functions

DmaChnOpen

Description: The function configures the selected DMA channel using the supplied user flags and priority.

Include: plib.h

Prototype: void DmaChnOpen(int chn, DmaChannelPri chPri, DmaOpenFlags oFlags);

Arguments:
chn The channel to be configured in the DMA controller.
chPri The priority given to the channel, 0-3.
oFlags orred flags specifying the open mode, as defined below:
   DMA_OPEN_NORM: DMA channel is to operate in normal mode
   DMA_OPEN_EXT: DMA channel is to operate in extended mode
   DMA_OPEN_AUTO: DMA channel is auto enabled
   DMA_OPEN_MATCH: DMA channel stops on pattern match

Return Value: None

Remarks: This is a high level access function that doesn't give access to all the settings possible for a DMA channel. Use the low level functions to address special settings.

After calling this function, the channel should be enabled using DmaChnEnable(chn) call.

If the CRC engine is attached to the submitted channel, the CRC append mode will be turned off. This way, the transfer will occur correctly together with CRC calculation.

The start and abort Iqrs will be disabled and the channel event enable flags are disabled. User has to call normal channel functions to enable the event flags if needed.

Source File: dma_chn_open_lib.c

Code Example: DmaChnOpen(3, DMA_CHN_PRI2, DMA_OPEN_EXT|DMA_OPEN_AUTO|DMA_OPEN_MATCH);

DmaChnEnable

Description: The function enables a previously configured DMA channel.

Include: plib.h

Prototype: void DmaChnEnable(int chn);

Arguments: chn The selected DMA channel.

Return Value: None
### DmaChnEnable

**Remarks:**
DmaChnOpen() should have been called before.

**Source File:**
dma_chn_enable_lib.c

**Code Example:**
DmaChnEnable(3);

### DmaChnDisable

**Description:**
The function disables a DMA channel. The channel operation stops.

**Include:**
plib.h

**Prototype:**
void DmaChnDisable(int chn);

**Arguments:**
chn The selected DMA channel.

**Return Value:**
None

**Remarks:**
DmaChnOpen() should have been called before.

**Source File:**
dma_chn_disable_lib.c

**Code Example:**
DmaChnDisable(3);

### DmaChnSetTxfer

**Description:**
The function sets the transfer characteristics for a normal (i.e. not extended) DMA channel transfer:
- the source and the destination addresses.
- the source and destination lengths
- and the number of bytes transferred per event.

**Include:**
plib.h

**Prototype:**
void DmaChnSetTxfer(int chn, const void* vSrcAdd, void* vDstAdd, int srcSize, int dstSize, int cellSize);

**Arguments:**
chn The selected DMA channel.
vSrcAdd source of the DMA transfer (virtual address)
vDstAdd destination of the DMA transfer (virtual address)
srcSize source buffer size, 1-256 bytes, wrapped around
dstSize destination buffer size, 1-256 bytes, wrapped around
cellSize cell transfer size, 1-256 bytes

**Return Value:**
None

**Remarks:**
None

**Source File:**
dma_chn_set_txfer_lib.c

**Code Example:**
DmaChnSetTxfer(3, &U2RXREG, dstBuff, 1, 256, 1);
DmaChnSetExtTxfer

Description: The function sets the transfer characteristics for an extended DMA channel transfer:
- the source and the destination addresses.
- the block transfer size.

Include: plib.h
Prototype: void DmaChnSetExtTxfer(int chn, const void* vSrcAdd, void* vDstAdd, int blockSize);
Arguments: chn The selected DMA channel.
vSrcAdd source of the DMA transfer (virtual address)
vDstAdd destination of the DMA transfer (virtual address)
blockSize block transfer size, 1-65536.

Return Value: None
Remarks: None
Source File: dma_chn_set_ext_txfer_lib.c
Code Example: DmaChnSetExtTxfer(3, srcBuff, dstBuff, 512);

DmaChnSetMatchPattern

Description: The function sets the current match pattern for the selected DMA channel.

Include: plib.h
Prototype: void DmaChnSetMatchPattern(int chn, int pattern);
Arguments: chn The selected DMA channel.
pattern the pattern to match for ending the DMA transfer

Return Value: None
Remarks: None
Source File: dma_chn_set_match_pattern_lib.c
Code Example: DmaChnSetMatchPattern(3, '\r');

DmaChnGetMatchPattern

Description: The function retrieves the current match pattern for the selected DMA channel.

Include: plib.h
Prototype: int DmaChnGetMatchPattern(int chn);
Arguments: chn The selected DMA channel.

Return Value: The channel match pattern.
Remarks: None
Source File: dma_chn_get_match_pattern_lib.c
Code Example: int stopPattern=DmaChnGetMatchPattern(3);
DmaChnStartTxfer

Description: The function enables the channel and initiates (forces) a DMA transfer for the selected DMA channel. If waiting for the transfer completion needed (user doesn't use an ISR to catch this event) the function will periodically query the DMA controller for the transfer completion status.

Include: plib.h

Prototype: DmaTxferRes DmaChnStartTxfer(int chn, DmaWaitMode wMode, unsigned long retries);

Arguments:
chn The selected DMA channel.
wMode The desired wait mode, as below:
   DMA_WAIT_NOT: return immediately
   DMA_WAIT_CELL: return after one cell transfer complete
   DMA_WAIT_BLOCK: return after the whole transfer is done
retries retry counter: if transfer not complete after so many retries, return with tmo. If 0, wait forever.

Return Value: DMA_TXFER_OK if not waiting for the transfer completion or if the transfer ended normally, an DmaTxferRes error code otherwise as below:
   DMA_TXFER_ADD_ERR: address error while performing the transfer
   DMA_TXFER_ABORT: the DMA transfer was aborted
   DMA_TXFER_BC_ERR: block complete not set after the DMA transfer performed
   DMA_TXFER_TMO: DMA transfer timeout

Remarks: This function can be used in both normal and extended mode. However, in extended mode there is no cell transfer, just block transfer. So DMA_WAIT_CELL wait mode is irrelevant.

Source File: dma_chn_start_txfer_lib.c
Code Example:
DmaTxferRes res = DmaChnStartTxfer(3, DMA_WAIT_BLOCK, 0);

DmaChnForceTxfer

Description: The function forces a DMA transfer to occur for the selected DMA channel.

Include: plib.h

Prototype: void DmaChnForceTxfer(int chn);

Arguments: chn The selected DMA channel.

Return Value: None

Remarks: None

Source File: dma_chn_force_txfer_lib.c
Code Example:
DmaChnForceTxfer(2);
### DmaChnAbortTxfer

**Description:** The function aborts a current undergoing DMA transfer for the selected DMA channel.

**Include:** plib.h

**Prototype:**

```c
void DmaChnAbortTxfer(int chn);
```

**Arguments:**

- `chn` The selected DMA channel.

**Return Value:** None

**Remarks:** None

**Source File:** dma_chn_abort_txfer_lib.c

**Code Example:**

```c
DmaChnAbortTxfer(2);
```

### DmaChnSetEvEnableFlags

**Description:** The function sets the event enable flags for the selected DMA channel. Multiple flags can be orr-ed together. Any flag that is set in the eFlags will be enabled for the selected channel, the other channel event flags won't be touched.

**Include:** plib.h

**Prototype:**

```c
void DmaChnSetEvEnableFlags(int chn, DmaEvFlags eFlags);
```

**Arguments:**

- `chn` The selected DMA channel.
- `eFlags` event flags with the following significance:
  - DMA_EV_ERR: address error event
  - DMA_EV_ABORT: transfer abort event
  - DMA_EV_CELL_DONE: cell transfer complete event
  - DMA_EV_BLOCK_DONE: block transfer complete event
  - DMA_EV_DST_HALF: destination half event
  - DMA_EV_DST_FULL: destination full event
  - DMA_EV_SRC_HALF: source half event
  - DMA_EV_SRC_FULL: source full event
  - DMA_EV_ALL_EVNTS: all of the above flags

**Return Value:** None

**Remarks:** None

**Source File:** dma_chn_set_ev_enable_flags_lib.c

**Code Example:**

```c
DmaChnSetEvEnableFlags(3, DMA_EV_ERR|DMA_EV_ABORT|DMA_EV_BLOCK_DONE|DMA_EV_SRC_FULL);
```
### DmaChnClrEvEnableFlags

**Description:** The function clears the event enable flags for the selected DMA channel. Multiple flags can be or-ed together. Any flag that is set in the eFlags will be enabled for the selected channel, the other channel event flags won’t be touched.

**Include:** `plib.h`

**Prototype:**
```c
void DmaChnClrEvEnableFlags(int chn, DmaEvFlags eFlags);
```

**Arguments:**
- `chn` The selected DMA channel.
- `eFlags` event flags with the following significance:
  - `DMA_EV_ERR`: address error event
  - `DMA_EV_ABORT`: transfer abort event
  - `DMA_EV_CELL_DONE`: cell transfer complete event
  - `DMA_EV_BLOCK_DONE`: block transfer complete event
  - `DMA_EV_DST_HALF`: destination half event
  - `DMA_EV_DST_FULL`: destination full event
  - `DMA_EV_SRC_HALF`: source half event
  - `DMA_EV_SRC_FULL`: source full event
  - `DMA_EV_ALL_EVNTS`: all of the above flags

**Return Value:** None

**Remarks:** None

**Source File:** `dma_chn_clr_ev_enable_flags_lib.c`

**Code Example:**
```c
DmaChnClrEvEnableFlags(3,
DMA_EV_ERR|DMA_EV_ABORT|DMA_EV_BLOCK_DONE|DMA_EV_SRC_FULL);
```

### DmaChnWriteEvEnableFlags

**Description:** The function sets the event enable flags for the selected DMA channel. The channel event flags are forced to the eFlags value.

**Include:** `plib.h`

**Prototype:**
```c
void DmaChnWriteEvEnableFlags(int chn, DmaEvFlags eFlags);
```

**Arguments:**
- `chn` The selected DMA channel.
- `eFlags` event flags with the following significance:
  - `DMA_EV_ERR`: address error event
  - `DMA_EV_ABORT`: transfer abort event
  - `DMA_EV_CELL_DONE`: cell transfer complete event
  - `DMA_EV_BLOCK_DONE`: block transfer complete event
  - `DMA_EV_DST_HALF`: destination half event
  - `DMA_EV_DST_FULL`: destination full event
```
DmaChnWriteEvEnableFlags

DMA_EV_SRC_HALF: source half event
DMA_EV_SRC_FULL: source full event
DMA_EV_ALL_EVNTS: all of the above flags

Return Value: None
Remarks: None
Source File: dma_chn_write_ev_enable_flags_lib.c
Code Example: DmaChnWriteEvEnableFlags(3, DMA_EV_ALL_EVNTS);

DmaChnGetEvEnableFlags

Description: The function returns the event enabled flags for the selected DMA channel.
Include: plib.h
Prototype: DmaEvFlags DmaChnGetEvEnableFlags(int chn);
Arguments: chn The selected DMA channel.
Return Value: event flags with the following significance:
  DMA_EV_ERR: address error event
  DMA_EV_ABORT: transfer abort event
  DMA_EV_CELL_DONE: cell transfer complete event
  DMA_EV_BLOCK_DONE: block transfer complete event
  DMA_EV_DST_HALF: destination half event
  DMA_EV_DST_FULL: destination full event
  DMA_EV_SRC_HALF: source half event
  DMA_EV_SRC_FULL: source full event
  DMA_EV_ALL_EVNTS: all of the above flags
Remarks: None
Source File: dma_chn_get_ev_enable_flags_lib.c
Code Example: DmaEvFlags enabledFlags=DmaChnGetEvEnableFlags(3);

DmaChnClrEvFlags

Description: The function clears the event flags for the selected DMA channel. Multiple flags can be or-ed together. Any flag that is set in the eFlags will be cleared for the selected channel, the other channel event flags won't be touched.
Include: plib.h
Prototype: void DmaChnClrEvFlags(int chn, DmaEvFlags eFlags);
Arguments: chn The selected DMA channel.
eFlags event flags with the following significance:
  DMA_EV_ERR: address error event
### DmaChnClrEvFlags

| Description | The function clears the event flags for the selected DMA channel.
| Include | plib.h
| Prototype | void DmaChnClrEvFlags(int chn);
| Arguments | chn  | The selected DMA channel.
| Return Value | None
| Remarks | None
| Source File | dma_chn_clr_ev_flags_lib.c
| Code Example | DmaChnClrEvFlags(3, DMA_EV_ALL_EVNTS); |

### DmaChnGetEvFlags

| Description | The function returns the current event flags for the selected DMA channel.
| Include | plib.h
| Prototype | DmaEvFlags DmaChnGetEvFlags(int chn);
| Arguments | chn  | The selected DMA channel.
| Return Value | event flags with the following significance:
| | DMA_EV_Err: address error event
| | DMA_EV_ABORT: transfer abort event
| | DMA_EV_CELL_DONE: cell transfer complete event
| | DMA_EV_BLOCK_DONE: block transfer complete event
| | DMA_EV_DST_HALF: destination half event
| | DMA_EV_DST_FULL: destination full event
| | DMA_EV_SRC_HALF: source half event
| | DMA_EV_SRC_FULL: source full event
| | DMA_EV_ALL_EVNTS: all of the above flags
| Remarks | None
| Source File | dma_chn_get_ev_flags_lib.c
| Code Example | DmaEvFlags enabledFlags=DmaChnGetEvFlags(3); |

### DmaChnIntEnable

| Description | The function enables the interrupts in the Interrupt Controller for the selected DMA channel.
| Source File | dma_chn_int_enable_lib.c
| Code Example | /* Enable interrupts for DMA Channel 3 */

| Description | The function/macro enables the interrupts in the Interrupt Controller for the selected DMA channel.
| Source File | dma_chn_int_enable_lib.c
| Code Example | /* Enable interrupts for DMA Channel 3 */

| Description | The function/macro enables the interrupts in the Interrupt Controller for the selected DMA channel.
| Source File | dma_chn_int_enable_lib.c
| Code Example | /* Enable interrupts for DMA Channel 3 */
**DmaChnIntEnable**

Include: `plib.h`

Prototype: `void DmaChnIntEnable(int chn);`

Arguments: `chn` The selected DMA channel.

Return Value: None

Remarks: None

Source File: `plib.h`

Code Example:

```c
int chn=3; DmaChnIntEnable(chn);
mDmaChnIntEnable(3);
```

**DmaChnIntDisable**

Include: `plib.h`

Prototype: `void DmaChnIntDisable(int chn);`

Arguments: `chn` The selected DMA channel.

Return Value: None

Remarks: None

Source File: `plib.h`

Code Example:

```c
int chn=3; DmaChnIntDisable(chn);
mDmaChnIntDisable(3);
```

**DmaChnGetIntEnable**

Include: `plib.h`

Prototype: `int DmaChnGetIntEnable(int chn);`

Arguments: `chn` The selected DMA channel.

Return Value: - TRUE if the corresponding interrupt is enabled, - FALSE otherwise

Remarks: None

Source File: `plib.h`

Code Example:

```c
int chn=3; int isEnabled=DmaChnGetIntEnable(chn);
int isEnabled=mDmaChnGetIntEnable(3);
```
DmaChnSetIntPriority
mDmaChnSetIntPriority

Description: The function/macro sets the interrupt priority and subpriority in the Interrupt Controller for the selected DMA channel.
Include: plib.h
Prototype: void DmaChnSetIntPriority(int chn, int iPri, int subPri);
Arguments: 
chn The selected DMA channel.
iPri the interrupt priority in the interrupt controller, 0-7.
subPri the interrupt subpriority in the interrupt controller, 0-3
Return Value: None
Remarks: None
Source File: plib.h
Code Example: int chn=0; DmaChnSetIntPriority(chn, INT_PRIORITY_LEVEL_5, INT_SUB_PRIORITY_LEVEL_3);
mDmaChnSetIntPriority(0, 5, 3);

DmaChnGetIntPriority
mDmaChnGetIntPriority

Description: The function/macro reads the current interrupt priority in the Interrupt Controller for the selected DMA channel.
Include: plib.h
Prototype: void DmaChnGetIntPriority(int chn);
Arguments: 
chn The selected DMA channel.
Return Value: None
Remarks: None
Source File: plib.h
Code Example: int chn=2; int currPri=DmaChnGetIntPriority(chn);
int currPri=mDmaChnGetIntPriority(2);

DmaChnGetIntSubPriority
mDmaChnGetIntSubPriority

Description: The function/macro reads the current interrupt sub priority in the Interrupt Controller for the selected DMA channel.
Include: plib.h
Prototype: void DmaChnGetIntSubPriority(int chn);
Arguments: 
chn The selected DMA channel.
Return Value: None
Remarks: None
Source File: plib.h
3.3 High level helpers for fast strcpy/memcpy transfers

DmaChnGetIntSubPriority
mDmaChnGetIntSubPriority

Code Example:

```
int chn=2; int currSPri = DmaChnGetIntSubPriority(chn);
int currSPri=mDmaChnGetIntSubPriority(2);
```

DmaChnGetIntFlag
mDmaChnGetIntFlag

Description: The function/macro reads the current interrupt flag in the Interrupt Controller for the selected DMA channel.

Include: plib.h

Prototype: int DmaChnGetIntFlag(int chn);

Arguments: chn The selected DMA channel.

Return Value: - TRUE if the corresponding channel interrupt flag is set
- FALSE otherwise

Remarks: None

Source File: plib.h

Code Example:

```
int chn=1; int isFlagSet=DmaChnGetIntFlag(chn);
isFlagSet=mDmaChnGetIntFlag(1);
```

DmaChnClrIntFlag
mDmaChnClrIntFlag

Description: The function/macro clears the interrupt flag in the Interrupt Controller for the selected DMA channel.

Include: plib.h

Prototype: void DmaChnClrIntFlag(int chn);

Arguments: chn The selected DMA channel.

Return Value: None

Remarks: None

Source File: plib.h

Code Example:

```
int chn=1; DmaChnClrIntFlag(chn);
mDmaChnClrIntFlag(1);
```
### DmaChnMemcpy

**Description:** The function copies one block of memory from source to destination.

**Include:**
```
plib.h
```

**Prototype:**
```
DmaTxferRes DmaChnMemcpy(void* s1, const void* s2, int n, int chn, DmaChannelPri chPri);
```

**Arguments:**
- `s1`: The destination pointer.
- `s2`: The source pointer.
- `n`: number of bytes to transfer, `n>0`, `n<=64K`
- `chn`: The DMA channel to perform the transfer
- `chPri`: The desired DMA channel priority, 0-3.

**Remarks:**
- If the CRC is attached to the submitted channel, the CRC append mode will be turned off. This way, the strcpy/memcpy transfers will occur correctly together with CRC calculation.
- The channel will operate in extended mode and will support transfers of up to 64K bytes.
- The start and abort IRQs will be disabled and the channel event enable flags, are disabled. User has to call normal channel functions to enable the event flags if needed.
- Multiple channels could be opened to perform fast memory transfers, if necessary.

**Return Value:**
- `DMA_TXFER_OK` if the transfer ended normally, a `DmaTxferRes` error code otherwise as below:
  - `DMA_TXFER_ADD_ERR`: address error while performing the transfer
  - `DMA_TXFER_ABORT`: the DMA transfer was aborted
  - `DMA_TXFER_BC_ERR`: block complete not set after the DMA transfer performed
  - `DMA_TXFER_TMO`: DMA transfer timeout

**Source File:**
```
dma_chn_memcpy_lib.c
```

**Code Example:**
```
DmaChnMemcpy(srcBuff, dstBuff, sizeof(dstBuff), 0, DMA_CHN_PRI3);
```

### DmaChnStrcpy

**Description:** The function copies one zero terminated string from source to destination.

**Include:**
```
plib.h
```

**Prototype:**
```
DmaTxferRes DmaChnStrcpy(char* s1, const char* s2, int chn, DmaChannelPri chPri);
```

**Arguments:**
- `s1`: The destination pointer.
- `s2`: The source pointer.
- `chn`: The DMA channel to perform the transfer
- `chPri`: The desired DMA channel priority, 0-3.

**Remarks:**
- If the CRC is attached to the submitted channel, the CRC append mode will be turned off. This way, the strcpy/memcpy transfers will occur correctly together with CRC calculation.
DmaChnStrcpy

The function copies one zero terminated string from source to destination. It copies no more than n characters from s2.

Include:
plib.h

Prototype:
DmaTxferRes DmaChnStrncpy(char* s1, const char* s2, int n, int chn, DmaChannelPri chPri);

Arguments:
s1 The destination pointer.
s2 The source pointer.
n max number of characters to be copied, n>0, n<=64K
chn The DMA channel to perform the transfer
chPri The desired DMA channel priority, 0-3.

Remarks:
If the CRC is attached to the submitted channel, the CRC append mode will be turned off. This way, the strcpy/memcpy transfers will occur correctly together with CRC calculation.

The channel will operate in extended mode and will support transfers of up to 64K bytes.

The start and abort Irqs will be disabled and the channel event enable flags are disabled. User has to call normal channel functions to enable the event flags if needed.

Multiple channels could be opened to perform fast memory transfers, if necessary.

Return Value:
DMA_TXFER_OK if the transfer ended normally, a DmaTxferRes error code otherwise as below:

DMA_TXFER_ADD_ERR: address error while performing the transfer
DMA_TXFER_ABORT: the DMA transfer was aborted
DMA_TXFER_BC_ERR: block complete not set after the DMA transfer performed
DMA_TXFER_TMO: DMA transfer timeout

Source File: dma_chn_strcpy_lib.c

Code Example: DmaChnStrcpy(str1, str2, 0, DMA_CHN_PRI3);

DmaChnStrncpy

Description:
The function copies one zero terminated string from source to destination. It copies no more than n characters from s2.

Include:
plib.h

Prototype:
DmaTxferRes DmaChnStrncpy(char* s1, const char* s2, int n, int chn, DmaChannelPri chPri);

Arguments:
s1 The destination pointer.
s2 The source pointer.
n max number of characters to be copied, n>0, n<=64K
chn The DMA channel to perform the transfer
chPri The desired DMA channel priority, 0-3.

Remarks:
If the CRC is attached to the submitted channel, the CRC append mode will be turned off. This way, the strcpy/memcpy transfers will occur correctly together with CRC calculation.

The channel will operate in extended mode and will support transfers of up to 64K bytes.

The start and abort Irqs will be disabled and the channel event enable flags are disabled. User has to call normal channel functions to enable the event flags if needed.

Multiple channels could be opened to perform fast memory transfers, if necessary.

Return Value:
DMA_TXFER_OK if the transfer ended normally, a DmaTxferRes error code otherwise as below:

DMA_TXFER_ADD_ERR: address error while performing the transfer
DMA_TXFER_ABORT: the DMA transfer was aborted
### DmaChnStrncpy

**Description:**
The function is a helper that calculates the CRC of a memory block.

**Include:**
plib.h

**Prototype:**
DmaTxferRes DmaChnStrncpy(const char* str1, const char* str2, int strLen, int chn, DmaChannelPri chPri);

**Arguments:**
- `str1`: Address of a variable where to deposit the result.
- `str2`: The start address of the memory area.
- `strLen`: Number of bytes in the memory area, n>0, n<=64K.
- `chn`: The DMA channel to perform the calculation.
- `chPri`: The desired DMA channel priority, 0-3.

**Remarks:**
The CRC generator must have been previously configured using mCrcConfigure().

**Return Value:**
- DMA_TXFER_OK if the transfer ended normally.
- A DmaTxferRes error code otherwise as below:

  - DMA_TXFER_ADD_ERR: address error while performing the transfer.
  - DMA_TXFER_ABORT: the DMA transfer was aborted.
  - DMA_TXFER_BC_ERR: block complete not set after the DMA transfer performed.
  - DMA_TXFER_TMO: DMA transfer timeout.

**Source File:**
dma_chn_strncpy_lib.c

**Code Example:**
DmaChnStrncpy(str1, str2, MAX_STR_LEN, 0, DMA_CHN_PRI3);

### DmaChnMemCrc

**Description:**
The function is a helper that calculates the CRC of a memory block.

**Include:**
plib.h

**Prototype:**
DmaTxferRes DmaChnMemCrc(void* d, const void* s, int n, int chn, DmaChannelPri chPri);

**Arguments:**
- `d`: Address of a variable where to deposit the result.
- `s`: The start address of the memory area.
- `n`: Number of bytes in the memory area, n>0, n<=64K.
- `chn`: The DMA channel to perform the calculation.
- `chPri`: The desired DMA channel priority, 0-3.

**Remarks:**
The CRC generator must have been previously configured using mCrcConfigure().

No transfer is done, just the CRC is calculated.

The channel will operate in extended mode and will support transfers of up to 64K bytes.

The start and abort Irqs will be disabled and the channel event enable flags, are disabled. User has to call normal channel functions to enable the event flags if needed.

Multiple channels could be opened to perform fast memory transfers, if necessary.

**Return Value:**
- DMA_TXFER_OK if the transfer ended normally, a DmaTxferRes error code otherwise as below:

  - DMA_TXFER_ADD_ERR: address error while performing the transfer.
  - DMA_TXFER_ABORT: the DMA transfer was aborted.
  - DMA_TXFER_BC_ERR: block complete not set after the DMA transfer performed.
  - DMA_TXFER_TMO: DMA transfer timeout.

**Source File:**
dma_chn_mem_crc_lib.c

**Code Example:**
int myCrc; DmaChnMemCrc(srcBuff, &myCrc, sizeof(srcBuff), 0, DMA_CHN_PRI3);

### 3.4 High level CRC functions
mCrcConfigure

Description: The macro configures the CRC module by setting the parameters that define the generator polynomial:
- the length of the CRC generator polynomial, pLen;
- the macro sets the layout of the shift stages that take place in the CRC generation.
Setting a bit to 1 enables the XOR input from the MSb (pLen bit) to the selected stage in the shift register. If bit is cleared, the selected shift stage gets data directly from the previous stage in the shift register. Note that in a proper CRC polynomial, both the most significant bit (MSb) and least significant bit (LSb) are always a '1'. Considering the generator polynomial: $X^{16}+X^{15}+X^2+1$, the value to be written as feedback should be 0x8005, or 0x8004, but not 0x018005;
- the macro sets the seed of the CRC generator. This is the initial data present in the CRC shift register before the CRC calculation begins. A good initial value is usually 0xffffffff.

Include: plib.h
Prototype: void mCrcConfigure(int polynomial, int pLen, int seed);
Arguments:
  polynomial The generator polynomial used for the CRC calculation.
  pLen the length of the CRC generator polynomial.
  seed the initial seed of the CRC generator.

Remarks: Bit 0 of the generator polynomial is always XOR'ed.
When the append mode is set, the attached DMA channel has to have destination size <=4. Upon the transfer completion the calculated CRC is stored at the destination address.
When append mode is cleared, the DMA transfer occurs normally, and the CRC value is available using the CrcResult() function.
The CRC module should be first configured and then enabled.

Return Value: None
Source File: plib.h
Code Example: mCrcConfigure(0x8005, 16, 0xffffffff);

CrcAttachChannel

Description: The function attaches the CRC module to an DMA channel and enables the CRC generator. From now on, all the DMA traffic is directed to the CRC generator. Once the DMA block transfer is complete, the CRC result is available both at the DMA destination address and in the CRC data register.

Include: plib.h
Prototype: void CrcAttachChannel(int chn, int appendMode);
Arguments: chn The DMA channel to be attached to the CRC generator module.
3.5 Low Level global DMA functions

### CrcAttachChannel

**appendMode** - if TRUE the data passed to the CRC generator is not transferred to destination but it's written to the destination address when the block transfer is complete.
- if FALSE the data is transferred normally while the CRC is calculated. The CRC will be available using the CrcResult function.

**Remarks:** None
**Return Value:** None
**Source File:** dma_crc_attach_channel_lib.c
**Code Example:** CrcAttachChannel(0, 1);

### CrcResult

**Description:** The function returns the calculated CRC value.
**Include:** plib.h
**Prototype:** int CrcResult(void);
**Arguments:** None
**Remarks:** The function returns the valid CRC result by masking out the unused MSbits in the CRC register. Use CrcGetValue() to get the full CRC register value.
**Return Value:** The current value of the CRC generator
**Source File:** dma_crc_result_lib.c
**Code Example:** int myCrc=CrcResult();

### mDmaEnable

**Description:** The macro enables the DMA controller.
**Include:** plib.h
**Prototype:** void mDmaEnable(void);
**Arguments:** None
**Return Value:** None
**Remarks:** None
**Source File:** dma.h
**Code Example:** mDmaEnable();
mDmaDisable

Description: The macro disables the DMA controller.
Include: plib.h
Prototype: void mDmaDisable(void);
Arguments: None
Return Value: None
Remarks: None
Source File: dma.h
Code Example: mDmaDisable();

mDmaReset

Description: The macro resets the DMA controller.
Include: plib.h
Prototype: void mDmaReset(void);
Arguments: None
Return Value: None
Remarks: None
Source File: dma.h
Code Example: mDmaReset();

mDmaSuspend

Description: The macro suspends the DMA controller activity. The activity can be later on resumed with mDmaResume();
Include: plib.h
Prototype: void mDmaSuspend(void);
Arguments: None
Return Value: None
Remarks: None
Source File: dma.h
Code Example: mDmaSuspend();

DmaGetStatus

Description: The function updates the info for the current DMA controller status. It updates the last DMA: operation, channel used and address.
Include: plib.h
Prototype: void DmaGetStatus(DmaStatus* pStat);
Arguments: pStat pointer to a DmaStatus structure to store the current DMA controller status, carrying the following info:
- chn: the last active DMA channel
- rdOp: the last DMA operation, read/write
DmaGetStatus

- lastAddress: the most recent DMA address

Return Value: None
Remarks: None
Source File: dma_get_status_lib.c
Code Example: DmaStatus stat; DmaGetStatus(&stat);

mDmaSetGlobalFlags

Description: The macro affects the global behavior of the DMA controller. It sets the specified flags. Any flag that is set in the gFlags will be enabled, the other flags won't be touched.
Include: plib.h
Prototype: void mDmaSetGlobalFlags(DmaGlblFlags gFlags);
Arguments: gFlags flags to be set, having the following fields:
- DMA_GFLG_SUSPEND: DMA controller operation suspend
- DMA_GFLG_SIDL: DMA controller sleep/active in idle mode
- DMA_GFLG_FRZ: DMA controller frozen/active in debug mode
- DMA_GFLG_ON: DMA controller enabled/desabled
- DMA_GFLG_ALL_FLAGS: all of the above flags
Remarks: None
Return Value: None
Source File: dma.h
Code Example: mDmaSetGlobalFlags(DMA_GFLG_SIDL|DMA_GFLG_ON);

mDmaClrGlobalFlags

Description: The macro affects the global behavior of the DMA controller. It clears the specified flags. Any flag that is set in the gFlags will be enabled, the other flags won't be touched.
Include: plib.h
Prototype: void mDmaClrGlobalFlags(DmaGlblFlags gFlags);
Arguments: gFlags flags to be cleared, having the following fields:
- DMA_GFLG_SUSPEND: DMA controller operation suspend
- DMA_GFLG_SIDL: DMA controller sleep/active in idle mode
Remarks: None
Return Value: None
Source File: dma.h
Code Example: mDmaClsrglobalFlags(DMA_GFLG_SIDL|DMA_GFLG_ON);
### mDmaClrGlobalFlags

**Description:** The macro affects the global behavior of the DMA controller. It forces the flags to have the specified `gFlags` value.

**Include:** `plib.h`

**Prototype:**

```c
void mDmaClrGlobalFlags(DmaGlblFlags gFlags);
```

**Arguments:**

- `gFlags` flags to be written, having the following fields:
  - `DMA_GFLG_SUSPEND`: DMA controller operation suspend
  - `DMA_GFLG_SIDL`: DMA controller sleep/active in idle mode
  - `DMA_GFLG_FRZ`: DMA controller frozen/active in debug mode
  - `DMA_GFLG_ON`: DMA controller enabled/desabled
  - `DMA_GFLG_ALL_FLAGS`: all of the above flags

**Remarks:** None

**Return Value:** None

**Source File:** `dma.h`

**Code Example:**

```c
mDmaClrGlobalFlags(DMA_GFLG_SIDL|DMA_GFLG_ON);
```

### mDmaWriteGlobalFlags

**Description:** The macro affects the global behavior of the DMA controller. It forces the flags to have the specified `gFlags` value.

**Include:** `plib.h`

**Prototype:**

```c
DmaGlblFlags mDmaGetGlobalFlags(void);
```

**Arguments:**

- `gFlags` flags to be written, having the following fields:
  - `DMA_GFLG_SUSPEND`: DMA controller operation suspend
  - `DMA_GFLG_SIDL`: DMA controller sleep/active in idle mode
  - `DMA_GFLG_FRZ`: DMA controller frozen/active in debug mode
  - `DMA_GFLG_ON`: DMA controller enabled/desabled
  - `DMA_GFLG_ALL_FLAGS`: all of the above flags

**Remarks:** None

**Return Value:** None

**Source File:** `dma.h`

**Code Example:**

```c
mDmaWriteGlobalFlags(DMA_GFLG_ALL_FLAGS);
```

### mDmaGetGlobalFlags

**Description:** The macro returns the global flags of the DMA controller.

**Include:** `plib.h`

**Prototype:**

```c
DmaGlblFlags mDmaGetGlobalFlags(void);
```

**Arguments:**

- `None`

**Remarks:** None

**Return Value:** The current DMA controller flags settings:

- `DMA_GFLG_SUSPEND`: DMA controller operation is suspended

**Source File:** `dma.h`
3.6 Low Level DMA channel status functions

mDmaGetGlobalFlags

- DMA_GFLG_SIDL: DMA controller sleep/active in idle mode
- DMA_GFLG_FRZ: DMA controller frozen/active in debug mode
- DMA_GFLG_ON: DMA controller enabled/desabled

Source File: dma.h
Code Example: DmaGlblFlags dmaFlags=mDmaGetGlobalFlags();

DmaChnGetSrcPnt

Description: The function retrieves the current source pointer for the selected DMA channel. In normal mode it is the current offset, 0-255, in the source transfer buffer.

Include: plib.h
Prototype: int DmaChnGetSrcPnt(int chn);
Arguments: chn The selected DMA channel
Remarks: This function is intended for use in normal mode. In the extended mode the source and destination pointers are concatenated into a 16 bit register. Use DmaChnGetDstPnt() instead.
Return Value: Current channel source pointer, 0-255
Source File: dma_chn_get_src_pnt_lib.c
Code Example: int srcPnt=DmaChnGetSrcPnt(3);

DmaChnGetDstPnt

Description: The function retrieves the current destination pointer for the selected DMA channel. In normal mode it is the current offset, 0-255, in the destination transfer buffer. In extended mode the function retrieves the current progress buffer pointer, ranging 0-65535.

Include: plib.h
Prototype: int DmaChnGetDstPnt(int chn);
Arguments: chn The selected DMA channel
Remarks: This function is intended for use in both normal and extended mode. In the extended mode the source and destination pointers are concatenated into a 16 bit register.
Return Value: Current channel destination pointer, 0-255 or 0-65535
Source File: dma_chn_get_dst_pnt_lib.c
Code Example: int dstPnt=DmaChnGetDstPnt(3);
3.7 Low Level DMA channel control functions

DmaChnGetCellPnt

Description: The function retrieves the current transfer progress pointer for the selected DMA channel. In normal mode it ranges 0-255.

Include: plib.h

Prototype: int DmaChnGetCellPnt(int chn);

Arguments: chn The selected DMA channel

Remarks: This function is intended for use in normal mode. There is no transfer pointer when in extended mode. Use DmaChnGetDstPnt().

Return Value: Current channel transfer pointer, 0-255.

Source File: dma_chn_get_cell_pnt_lib.c

Code Example: int cellPnt=DmaChnGetCellPnt(3);

DmaChnSetEventControl

Description: The function sets the events that start and abort the transfer for the selected DMA channel.

Include: plib.h

Prototype: void DmaChnSetEventControl(int chn, unsigned int dmaEvCtrl);

Arguments: chn The selected DMA channel
dmaEvCtrl flags controlling the DMA events, as below:
- DMA_EV_ABORT_IRQ_EN: enable/disable the abort IRQ action
- DMA_EV_START_IRQ_EN: enable/disable the start IRQ action
- DMA_EV_MATCH_EN: enable/disable the pattern match and abort
- DMA_EV_START_IRQ(irq): IRQ number to start the DMA channel transfer
- DMA_EV_ABORT_IRQ(irq): IRQ number to abort the DMA channel transfer

Remarks: None

Return Value: None

Source File: dma_chn_set_event_control_lib.c

Code Example: DmaChnSetEventControl(3, DMA_EV_START_IRQ_EN|DMA_EV_MATCH_EN|DMA_EV_START_IRQ(_UART2_RX_IRQ));
DmaEvCtrl evCtrl; evCtrl.w=0; evCtrl.abortIrqEn=1; evCtrl.matchEn=1; evCtrl.startIrq=_UART2_RX_IRQ;
DmaChnSetEventControl(3, evCtrl.w);
**DmaChnGetEventControl**

**Description:** The function retrieves the events that start and abort the transfer for the selected DMA channel.

**Include:** plib.h

**Prototype:**

```c
unsigned int DmaChnGetEventControl(int chn);
```

**Arguments:**

- `chn` The selected DMA channel

**Remarks:** None

**Return Value:** the current flags controlling the DMA events, as below:

- **DMA_EV_ABORT_IRQ_EN:** enable/disable the abort IRQ action
- **DMA_EV_START_IRQ_EN:** enable/disable the start IRQ action
- **DMA_EV_MATCH_EN:** enable/disable the pattern match and abort
- **DMA_EV_START_IRQ(irq):** IRQ number to start the DMA channel transfer
- **DMA_EV_ABORT_IRQ(irq):** IRQ number to abort the DMA channel transfer

**Source File:** dma_chn_get_event_control_lib.c

**Code Example:**

```c
int evCtrlW=DmaChnGetEventControl(3);
if(evCtrlW&DMA_EV_MATCH_EN) {...}
DmaEvCtrl evCtrl; evCtrl.w=DmaChnGetEventControl(3);
if(evCtrl.matchEn){...}
```

---

**DmaChnSetControl**

**Description:** The function enables/disables the selected DMA channel and also sets the channel priority, chaining mode, auto or extended mode and events detection.

**Include:** plib.h

**Prototype:**

```c
void DmaChnSetControl(int chn, unsigned int dmaChnCtrl);
```

**Arguments:**

- `chn` The selected DMA channel
- `dmaChnCtrl` any of the DMA channel control flags:
  - **DMA_CTL_PRI(pri):** channel priority 0-3
  - **DMA_CTL_EXT_EN:** enable/disable the extended mode
  - **DMA_CTL_AUTO_EN:** enable/disable the automatic mode
  - **DMA_CTL_CHAIN_EN:** enable/disable channel chaining
  - **DMA_CTL_DET_EN:** enable/disable events detection when channel disabled
  - **DMA_CTL_CHN_EN:** enable/disable channel functionality
  - **DMA_CTL_CHAIN_DIR:** chain direction: chain to lower(1)/higher(0), pri channel

**Remarks:** None
**DmaChnSetControl**

**Return Value:** None

**Source File:** dma_chn_set_control_lib.c

**Code Example:**

```c
DmaChnSetControl(3,
    DMA_CTL_PRI(DMA_CHN_PRI2)|DMA_CTL_AUTO_EN|DMA_CTL_CHAIN_EN);
DmaChnCtrl chCtrl; chCtrl.w=0;
chCtrl.chPri=DMA_CHN_PRI2; chCtrl.autoEn=1;
chCtrl.chainEn=1; DmaChnSetControl(3, chCtrl.w);
```

---

**DmaChnGetControl**

**Description:** The function retrieves the current control settings for the selected DMA channel, including the channel enable/disable status, the channel priority, chaining mode, auto or extended mode and events detection.

**Include:** plib.h

**Prototype:**

```c
unsigned int DmaChnGetControl(int chn);
```

**Arguments:**

- `chn`: The selected DMA channel

**Remarks:** None

**Return Value:** DMA channel control flags as follows:

- DMA_CTL_PRI(pri): channel priority 0-3
- DMA_CTL_EXT_EN: enable/disable the extended mode
- DMA_CTL_AUTO_EN: enable/disable the automatic mode
- DMA_CTL_CHAIN_EN: enable/disable channel chaining
- DMA_CTL_DET_EN: enable/disable events detection when channel disabled
- DMA_CTL_CHAN_EN: enable/disable channel functionality
- DMA_CTL_CHAIN_DIR: chain direction: chain to lower(1)/higher(0), pri channel

**Source File:** dma_chn_get_control_lib.c

**Code Example:**

```c
unsigned int ctrl=DmaChnGetControl(3);
if(ctrl&DMA_CTL_AUTO_EN) {...}
DmaChnCtrl chnCtrl; chnCtrl.w=DmaChnGetControl(3);
if(chnCtrl.autoEn) {...}
```

---

**DmaChnGetEvDetect**

**Description:** The function returns the current event detection setting for the selected DMA channel.

**Include:** plib.h

**Prototype:**

```c
int DmaChnGetEvDetect(int chn);
```

**Arguments:**

- `chn`: The selected DMA channel

**Source File:** dma_chn_get_ev_detect_lib.c

**Code Example:**

```c
DmaChnGetEvDetect(3);
if(ctrl&DMA_CTL_AUTO_EN) {...}
```
3.8 Low Level CRC control functions

mCrcEnable

Description: The macro enables the CRC module functionality and the attached DMA channel transfers are routed to the CRC module.
mCrcEnable

Include: plib.h
Prototype: void mCrcEnable(void);
Arguments: None
Remarks: None
Return Value: None
Source File: dma.h
Code Example: mCrcEnable();

mCrcDisable

Description: The macro disables the CRC module functionality.
Include: plib.h
Prototype: void mCrcDisable(void);
Arguments: None
Remarks: None
Return Value: None
Source File: dma.h
Code Example: mCrcDisable();

mCrcGetEnable

Description: The macro returns the CRC module enabling status.
Include: plib.h
Prototype: int mCrcGetEnable(void);
Arguments: None
Remarks: None
Return Value: - TRUE, if the CRC module is enabled
- FALSE otherwise
Source File: dma.h
Code Example: int isCrcEnabled=mCrcGetEnable();

mCrcAppendModeEnable

Description: The macro enables the CRC append mode. In this mode, the attached
DMA channel reads the source data but does not write it to the
destination address. The data it's just passed to the CRC generator for
CRC calculation. When the block transfer is completed, the CRC result
is written to the DMA channel destination address.
Include: plib.h
Prototype: void mCrcAppendModeEnable();
Arguments: None
Remarks: The CRC module should be properly configured before enabled.
mCrcAppendModeEnable

Description: The macro enables the CRC append mode. When the append mode is enabled, the attached DMA channel normally transfers data from source to destination. Data is also passed to the CRC controller for CRC calculation. When the DMA transfer is completed, the CRC value is available using the CrcGetValue function.

Include: plib.h
Prototype: void mCrcAppendModeEnable();
Arguments: None
Remarks: None.
Return Value: None
Source File: dma.h
Code Example: mCrcAppendModeEnable();

mCrcAppendModeDisable

Description: The macro disables the CRC append mode. When the append mode is disabled, the attached DMA channel normally transfers data from source to destination. Data is also passed to the CRC controller for CRC calculation. When the DMA transfer is completed, the CRC value is available using the CrcGetValue function.

Include: plib.h
Prototype: void mCrcAppendModeDisable();
Arguments: None
Remarks: None.
Return Value: None
Source File: dma.h
Code Example: mCrcAppendModeDisable();

mCrcGetAppendMode

Description: The macro returns the current CRC module enabling status.

Include: plib.h
Prototype: int mCrcGetAppendMode(void);
Arguments: None
Remarks: None.
Return Value: - TRUE, if the CRC append mode is enabled
             - FALSE otherwise
Source File: dma.h
Code Example: int isAppendEnabled=mCrcGetAppendMode();

mCrcSetDmaAttach

Description: The macro attaches a DMA channel to the CRC module. The DMA channel transfers will be routed to the CRC module.

Include: plib.h
Prototype: void mCrcSetDmaAttach(int chn);
Arguments: chn The selected DMA channel to be attached
Remarks: None
Return Value: None
Source File: dma.h
Code Example: mCrcSetDmaAttach(3);
### mCrcGetDmaAttach

**Description:** The macro returns the DMA channel number that is currently attached to the CRC module.

**Include:** plib.h

**Prototype:**

```c
int mCrcGetDmaAttach(void);
```

**Arguments:** None

**Remarks:** None

**Return Value:** The DMA channel that is currently attached to the CRC module

**Source File:** dma.h

**Code Example:**

```c
int chn=mCrcGetDmaAttach();
```

### mCrcSetPLen

**Description:** The macro sets the length of the CRC generator polynomial;

**Include:** plib.h

**Prototype:**

```c
void mCrcSetPLen(int pLen);
```

**Arguments:**

- `pLen` - the length of the CRC generator polynomial

**Remarks:** None

**Return Value:** None

**Source File:** dma.h

**Code Example:**

```c
mCrcSetPLen(16);
```

### mCrcGetPLen

**Description:** The macro returns the length of the CRC generator polynomial;

**Include:** plib.h

**Prototype:**

```c
int mCrcGetPLen(void);
```

**Arguments:** None

**Remarks:** None

**Return Value:** The length of the CRC generator polynomial

**Source File:** dma.h

**Code Example:**

```c
int polyLen=mCrcGetPLen();
```

### mCrcSetShiftFeedback

**Description:** The macro sets the layout of the shift stages that take place in the CRC generation. Setting a bit to 1 enables the XOR input from the MSb (pLen bit) to the selected stage in the shift register. If bit is cleared, the selected shift stage gets data directly from the previous stage in the shift register.
### mCrcSetShiftFeedback

**Include:** plib.h  
**Prototype:** void mCrcSetShiftFeedback(int feedback);  
**Arguments:**  
- `feedback` The layout of the CRC generator (shift register)  
**Remarks:** Bit 0 of the generator polynomial is always XOR'ed.  
**Return Value:** None  
**Source File:** dma.h  
**Code Example:** mCrcSetShiftFeedback(0x8005);

### mCrcGetShiftFeedback

**Description:** The macro returns the layout of the shift stages that take place in the CRC generation. A bit set to 1 enables the XOR input from the MSb (pLen bit) to the selected stage in the shift register. If a bit is cleared, the selected shift stage gets data directly from the previous stage in the shift register.  
**Include:** plib.h  
**Prototype:** int mCrcGetShiftFeedback(void);  
**Arguments:** None  
**Remarks:** Bit 0 of the generator polynomial is always XOR'ed.  
**Return Value:** The current layout of the CRC generator (shift register).  
**Source File:** dma.h  
**Code Example:** int feedback=mCrcGetShiftFeedback();

### mCrcSetSeed

**Description:** The macro sets the seed of the CRC generator. This is the initial data present in the CRC shift register before the CRC calculation begins.  
**Include:** plib.h  
**Prototype:** void mCrcSetSeed(int seed);  
**Arguments:**  
- `seed` The initial seed of the CRC generator  
**Remarks:** None  
**Return Value:** None  
**Source File:** dma.h  
**Code Example:** mCrcSetSeed(0xffff);

### mCrcGetValue

**Description:** The macro returns the current value of the CRC shift register.  
**Include:** plib.h  
**Prototype:** int mCrcGetValue(void);  
**Arguments:** None
3.9 Channel test/debug and special functions

**DmaChnSetEvFlags**

**Description:** The function sets the event flags for the selected DMA channel. Multiple flags can be or-ed together. Any flag that is set in the eFlags will be set for the selected channel, the other channel event flags won't be touched.

**Include:** plib.h

**Prototype:**

```c
void DmaChnSetEvFlags(int chn, DmaEvFlags eFlags);
```

**Arguments:**

- `chn` This is the number of the DMA channel
- `eFlags` event flags with the following significance:
  - `DMA_EV_ERR`: address error event
  - `DMA_EV_ABORT`: transfer abort event
  - `DMA_EV_CELL_DONE`: cell transfer complete event
  - `DMA_EV_BLOCK_DONE`: block transfer complete event
  - `DMA_EV_DST_HALF`: destination half event
  - `DMA_EV_DST_FULL`: destination full event
  - `DMA_EV_SRC_HALF`: source half event
  - `DMA_EV_SRC_FULL`: source full event
  - `DMA_EV_ALL_EVNTS`: all of the above flags

**Remarks:** This is intended as a channel test function.

**Return Value:** None

**Source File:** dma_chn_set_ev_flags_lib.c

**Code Example:**

```c
DmaChnSetEvFlags(0, DMA_EV_ERR|DMA_EV_ABORT|DMA_EV_BLOCK_DONE|DMA_EV_SRC_FULL);
```

**DmaChnWriteEvFlags**

**Description:** The function writes the event flags for the selected DMA channel. The channel event flags are forced to the eFlags value.

**Include:** plib.h

**Prototype:**

```c
void DmaChnWriteEvFlags(int chn, DmaEvFlags eFlags);
```

**Arguments:**

- `chn` This is the number of the DMA channel

**Source File:** dma_chn_set_ev_flags_lib.c

**Code Example:**

```c
DmaChnSetEvFlags(0, DMA_EV_ERR|DMA_EV_ABORT|DMA_EV_BLOCK_DONE|DMA_EV_SRC_FULL);
```
### DmaChnWriteEvFlags

**Description:**
The function sets the DMA controller behavior in Debug mode. It allows writing event flags with the following significance:

- **DMA_EV_ERR**: address error event
- **DMA_EV_ABORT**: transfer abort event
- **DMA_EV_CELL_DONE**: cell transfer complete event
- **DMA_EV_BLOCK_DONE**: block transfer complete event
- **DMA_EV_DST_HALF**: destination half event
- **DMA_EV_DST_FULL**: destination full event
- **DMA_EV_SRC_HALF**: source half event
- **DMA_EV_SRC_FULL**: source full event
- **DMA_EV_ALL_EVNTS**: all of the above flags

**Remarks:**
This is intended as a channel test function.

**Return Value:** None

**Source File:** dma_chn_write_ev_flags_lib.c

**Code Example:**
```c
DmaChnWriteEvFlags(0,
     DMA_EV_ERR|DMA_EV_ABORT|DMA_EV_BLOCK_DONE|DMA_EV_SRC_FULL)
```

---

### mDmaFreezeEnable

**Description:**
The macro sets the DMA controller behavior in Debug mode. The DMA controller is frozen in Debug mode.

**Include:**
plib.h

**Prototype:**
```c
void mDmaFreezeEnable();
```

**Arguments:** None

**Remarks:**
This macro is intended to be used in a debug handler.

**Return Value:** None

**Source File:** dma.h

**Code Example:**
mDmaFreezeEnable();

---

### mDmaFreezeDisable

**Description:**
The macro sets the DMA controller behavior in Debug mode. The DMA controller continues to run in Debug mode.

**Include:**
plib.h

**Prototype:**
```c
void mDmaFreezeDisable();
```

**Arguments:** None

**Remarks:**
This macro is intended to be used in a debug handler.

**Return Value:** None

**Source File:** dma.h

**Code Example:**
mDmaFreezeDisable();
### 3.10 Very low level access functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Include</th>
<th>Prototype</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>DmaChnSetRegister</td>
<td>The function sets directly a value into a DMA channel register.</td>
<td>plib.h</td>
<td>void DmaChnSetRegister(int chn, DmaChnRegIx regIx, int value);</td>
<td>chn  This is the number of the DMA channel regIx  register index, having one of the following enumerated values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_CON: control register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_CON_CLR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_CON_SET</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_CON_INV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_ECON: event control register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_ECON_CLR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_ECON_SET</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_ECON_INV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_INTR: interrupt control register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_INTR_CLR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_INTR_SET</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_INTR_INV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_SSA: source address register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_SSA_CLR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_SSA_SET</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_SSA_INV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_DSA: destination address register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_DSA_CLR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_DSA_SET</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_DSA_INV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_SSIZ: source size register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_SSIZ_CLR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_SSIZ_SET</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_SSIZ_INV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_DSIZ: destination size register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_DSIZ_CLR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_DSIZ_SET</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_DSIZ_INV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_SPTR: source pointer register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_DPTR: destination pointer register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA_REG_IX_CSIZ: cell size register</td>
</tr>
</tbody>
</table>

© 2007 Microchip Technology Inc.
### DmaChnSetRegister

<table>
<thead>
<tr>
<th>Description:</th>
<th>DMA_REG_IX_CSIZ_CLR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMA_REG_IX_CSIZ_SET</td>
</tr>
<tr>
<td></td>
<td>DMA_REG_IX_CSIZ_INV</td>
</tr>
<tr>
<td>DMA_REG_IX_CPTR: cell pointer register</td>
<td>DMA_REG_IX_DAT_CLR</td>
</tr>
<tr>
<td>DMA_REG_IX_DAT: pattern data register</td>
<td>DMA_REG_IX_DAT_SET</td>
</tr>
<tr>
<td>DMA_REG_IX_DAT_INV</td>
<td>value value to be written to the register.</td>
</tr>
</tbody>
</table>

**Remarks:** This is intended as a low level access channel function.

**Return Value:** None

**Source File:** dma_chn_set_register_lib.c

**Code Example:**

```c
DmaChnSetRegister(3, DMA_REG_IX_SSIZ, myBuffSz);
```

### DmaChnGetRegister

<table>
<thead>
<tr>
<th>Description:</th>
<th>The function retrieves the current value of a DMA channel register.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>plib.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td>int DmaChnGetRegister(int chn, DmaChnRegIx regIx);</td>
</tr>
<tr>
<td>Arguments:</td>
<td>chn This is the number of the DMA channel</td>
</tr>
<tr>
<td></td>
<td>regIx register index, having one of the following enumerated values:</td>
</tr>
<tr>
<td></td>
<td>DMA_REG_IX_CON: control register</td>
</tr>
<tr>
<td></td>
<td>DMA_REG_IX_ECON: event control register</td>
</tr>
<tr>
<td></td>
<td>DMA_REG_IX_INTR: interrupt control register</td>
</tr>
<tr>
<td></td>
<td>DMA_REG_IX_SSA: source address register</td>
</tr>
<tr>
<td></td>
<td>DMA_REG_IX_DSA: destination address register</td>
</tr>
<tr>
<td></td>
<td>DMA_REG_IX_SSIZ: source size register</td>
</tr>
<tr>
<td></td>
<td>DMA_REG_IX_DSIZ: destination size register</td>
</tr>
<tr>
<td></td>
<td>DMA_REG_IX_SPTR: source pointer register</td>
</tr>
<tr>
<td></td>
<td>DMA_REG_IX_DPTR: destination pointer register</td>
</tr>
<tr>
<td></td>
<td>DMA_REG_IX_CSIZ: cell size register</td>
</tr>
<tr>
<td>Remarks:</td>
<td>This is intended as a low level access channel function.</td>
</tr>
<tr>
<td></td>
<td>Read from CLR/SET/INV registers yields undefined value.</td>
</tr>
<tr>
<td>Return Value:</td>
<td>The current register value.</td>
</tr>
<tr>
<td>Source File:</td>
<td>dma_chn_set_register_lib.c</td>
</tr>
<tr>
<td>Code Example:</td>
<td>unsigned int mySrcSizeReg=DmaChnGetRegister(3, DMA_REG_IX_SSIZ);</td>
</tr>
</tbody>
</table>
3.11 Example of Use

Example 1: a CRC calculation.

```c
#include <plib.h>

// configuration settings
#pragma config POSCMOD = HS, FNOSC = PRIPLL
#pragma config PLLMUL = MUL_18, PLLIDIV = DIV_2
#pragma config FWDTEN = OFF

int main(void)
{
    // configure the proper PB frequency and the number of wait states
    SYSTEMConfigWaitStatesAndPB(72000000L);
    CheKeq0CacheOn();// enable the cache for the best performance
    mBMXSetArbMode(2);// arbitration mode 2, round-robin

    // first we'll show how to calculate CRC using the DMA controller
    {
        #define CRC_BUFF_SIZE 2048// the size of the memory area for
        // which to calculate the CRC
        unsigned char*romBuff=(unsigned char*)0xbfc00000;
        // we use the BOOT Flash to calculate its CRC
        unsigned inthwCrc;// we're going to calculate the CRC
        // and deposit here
        int chn=2; // DMA channel to use for our example
        DmaTxferResres;

        // we'll use the standard CCITT CRC 16 polynomial:
        // X^16+X^12+X^5+1, hex=0x00011021
        // calculate the CRC of the FLASH area. No DMA transfer occurs.
        // we use the high level method exposed by the DMA API

        // before using the DmaChnMemCrc() function,
        // the CRC has to be initialized:
        mCrcConfigure(0x11021, 16, 0xffff); // seed set to 0xffff

        res=DmaChnMemCrc(&hwCrc, romBuff, CRC_BUFF_SIZE, chn, DMA_CHN_PRI2);
        if(res!=DMA_TXFER_OK)
        {
            return0;   // DMA calculation failed
        }
        // we have now the CRC available in the hwCrc variable
        // and we can use it.
        // CRC calculation done successfully
    }
    return 1;
}
```

Example 2: a memory to memory copy.

```c
#include <stdlib.h>
#include <plib.h>

// configuration settings
#pragma config FNOSC = PRIPLL, POSCMOD = HS
#pragma config PLLMUL = MUL_18, PLLIDIV = DIV_2
#pragma config PWRTEN = OFF
#pragma config FWDTEN = OFF, WINDIS = OFF, WDTPS = PS1
#pragma config FCKSM = CSDCMD
```
```c
#pragma config OSCIOFNC = OFF
#pragma config IESO = OFF, LPOSCEN = OFF, GCP = OFF
#pragma config BWF = OFF, PWF = OFF, ICESEL = ICS_PGx1, BKBUG = OFF

int main(void)
{

    SYSTEMConfigWaitStatesAndPB(72000000L);
    // configure the proper PB frequency and the number of wait states
    CheKseg0CacheOn(); // enable the cache for the best performance
    mBXMSetArbMode(2); // arbitration mode 2, round-robin

    // a memory to memory copy
    #define MIN_RAM_TXFER_SIZE 8 // min size per transfer
    #define MAX_RAM_TXFER_SIZE 512 // we test the extended mode when
    // transfer more than 256 bytes

    unsigned char *pDmaSrc;
    unsigned char *pDmaDst;
    unsigned int txferSize;
    DmaTxferRestxferRes;
    DmaOpenFlags oFlag; // DMA open flags
    int dmaOk = 0;
    int matchOk = 0;
    int allocOk = 0; // operations ok flags
    int chn = 3; // DMA channel to use for our example

    srand((int)__TIME__); // seed the pseudo random generator

    txferSize = MIN_RAM_TXFER_SIZE
    txferSize += rand() % (MAX_RAM_TXFER_SIZE - MIN_RAM_TXFER_SIZE + 1);
    // get a random transfer size

    oFlag = txferSize > 256? DMA_OPEN_EXT : DMA_OPEN_NORM;

    DmaChnOpen(chn, DMA_CHN_PRI2, oFlag);
    // configure the DMA controller appropriately

    pDmaSrc = (unsigned char*)malloc(txferSize);
    pDmaDst = (unsigned char*)malloc(txferSize);

    if (pDmaSrc && pDmaDst)
    {
        unsigned char *pS;
        unsigned char *pD;
        int ix;
        allocOk = 1;

        for (ix = 0, pS = pDmaSrc; ix < txferSize; ix++)
        {
            *pS++ = rand(); // fill the source buffer
        }

        if (txferSize > 256)
        {
            // extended mode transfer
            // program the DMA channel source add, dest add, block size
            DmaChnSetExtTxfer(chn, pDmaSrc, pDmaDst, txferSize);
        }
        else
        {
            // normal mode transfer
            // program the DMA channel source add, dest add,
            // source and dest size, cell size adjusted
        }
    }

    return 0;
}
```
DmaChnSetTxfer(chn, pDmaSrc, pDmaDst, txferSize, txferSize, txferSize);
}

// start the DMA transfer and wait for it to finish
txferRes=DmaChnStartTxfer(chn, DMA_WAIT_BLOCK, 0);
if(txferRes==DMA_TXFER_OK)
{
    dmaOk=1;
    matchOk=1;
    for(ix=0, pS=pDmaSrc, pD=pDmaDst; ix<txferSize; ix++)
    {
        if(*pS++!=*pD++)
        {
            matchOk=0;
            break;
        }
    }
}
free(pDmaDst);
free(pDmaSrc);
return dmaOk && matchOk && allocOk;
# 4.0 BUS MATRIX FUNCTIONS

This section contains a list of macros for Bus Matrix.

## 4.1 Individual Functions/Macros

### mBMXSetArbMode

**Description:** This macro sets the bus matrix arbitration mode in BMXCON register.

**Include:** plib.h

**Prototype:**

```c
mBMXSetArbMode(mode)
```

**Arguments:**

- `mode` - mode = 0, 1 or 2

**Return Value:** None

**Remarks:**

- Code Example:
  ```c
  mBMXSetArbMode(1); //set arb mode to 1
  ```

### mBMXEnableBreakExactDRM / mBMXDisableBreakExactDRM

**Description:** this macro enables and disables break-exact debug mode

**Include:** plib.h

**Prototype:**

```c
mBMXEnableBreakExactDRM(), mBMXDisableBreakExactDRM()
```

**Arguments:** None

**Return Value:** None

**Remarks:**

- Code Example:
  ```c
  mBMXEnableBreakExactDRM();
  ```

### mBMXEnableXcpts / mBMXDisableXcpts

**Description:** this macro enables and disables exception generation by BMX

**Include:** plib.h

**Prototype:**

```c
mBMXEnableXcpts(val), mBMXDisableXcpts(val)
```

**Arguments:** Exception bit position in BMXCON register:

- `BMX_IXI_XCPT`
- `BMX_ICD_XCPT`
- `BMX_DMA_XCPT`
- `BMX_DS_XCPT`
- `BMX_IS_XCPT`

**Return Value:** None

**Remarks:**

- Code Example:
  ```c
  mBMXEnableXcpts(BMX_DS_XCPT); //enable data side bus error exceptions.
  ```

### mSetFlashPartition

**Description:** This macro sets the Flash Partition sizes
### mSetFlashUserPartition

**Include:** plib.h  
**Prototype:** `mSetFlashUserPartition (USER_FLASH_PGM_SZ)`  
**Arguments:** `USER_FLASH_PGM_SZ` - Partition Size in Bytes for user mode Program in Flash  
**Return Value:** None  
**Remarks:** The macro initializes the Base Address registers for partitioning the on-chip Flash. The Flash memory is divided into two partitions. By default the entire Flash is mapped to Kernel mode program space. If this macro is called with a non-zero value, the total Flash size minus this value (user mode program size) is assigned to the Kernel mode program space in Flash.  
**Code Example:**  
Example:  
mBMXSetFlashUserPartition(0x2000);  //set user mode program partition in flash to 8KBytes

### mBMXSetRAMKernProgOffset

**Description:** This macro sets the BMXDPBA register  
**Include:** plib.h  
**Prototype:** `mBMXSetRAMKernProgOffset (offset)`  
**Arguments:** `offset` - Offset into the RAM for start of Kernel Program partition  
**Return Value:** None  
**Remarks:** To execute code from RAM, the BMXDPBA must be set properly. This macro initializes this register.  
**Code Example:**  
Example:  
mBMXSetRAMKernProgOffset(0x4000);  //set kernel prog start at 0x4000 in RAM

### mBMXSetRAMUserDataOffset

**Description:** This macro sets the BMXDUPBA register  
**Include:** plib.h  
**Prototype:** `mBMXSetRAMUserDataOffset (offset)`  
**Arguments:** `offset` - Offset into the RAM for start of user data partition  
**Return Value:** None  
**Remarks:** For user mode data RAM, the BMXDUPBA must be set properly. This macro initializes this register.  
**Code Example:**  
Example:  
mBMXSetRAMUserDataOffset(0x6000);  //set user-mode data to start at 0x6000 in RAM

### mBMXSetRAMUserProgOffset

**Description:** This macro sets the BMXDUPBA register  
**Include:** plib.h
### mBMXSetRAMUserProgOffset

**Prototype:**

```c
mBMXSetRAMUserProgOffset (offset)
```

**Arguments:**

- `offset`: Offset into the RAM for start of user-mode Program partition

**Return Value:**

None

**Remarks:**

To execute code from RAM in user mode, the BMXDUPBA must be set properly. This macro initializes this register.

**Code Example:**

```c
mBMXSetRAMUserProgOffset(0x7000); //set kernel prog start at 0x8000 in RAM
```

### BMXCON Bit Set/Clear macros

**Description:**

These macros set/clear individual bits in the BMXCON register.

**Include:**

`plib.h`

**Prototype:**

```c
mBMXEnableIxiExpt
mBMXDisableIxiExpt
mBMXEnableCpuDExpt
mBMXDisableCpuDExpt
mBMXEnableCpuIExpt
mBMXDisableCpuIExpt
mBMXEnablePfmCheDma
mBMXDisablePfmCheDma
```

**Arguments:**

None

**Return Value:**

None

**Remarks:**

These macros let the programmer change individual bits for exception generation in the bus matrix config BMXCON register. This method allows the programmer to change the required bit without effecting the rest of the configuration bits.

**Code Example:**

```c
... ...
mBMXEnableDmaExpt(); //Turn on DMA unmapped address exceptions ...
... ...
ShutDown:
...
...
mBMXDisableDmaExpt(); // Turn off exceptions ...
```
5.0 NVM FUNCTIONS

This section contains a list of individual functions for NVM and an example of use of the functions.

5.1 Individual Functions

NVMProgram

Description: This function programs size characters from the source buffer to Flash memory starting at the destination address.

Include: plib.h

Prototype: unsigned int NVMProgram( void *address, const void *data, unsigned int size, void * pagebuff )

Arguments:
*address Pointer to destination virtual address to start writing from.
*data Pointer to source data to write.
size Number of bytes to write.
*pagebuff Working page buffer in RAM

Return Value: '0' if operation completed successfully

Remarks: None

Code Example:
NVMProgram((void*) 0xBD000000, (const void*) 0xA0000000, 1024, (void *) 0xA0002000);

NVMErasePage

Description: This function erases a single page of program flash.

Include: plib.h

Prototype: unsigned int NVMErasePage(void* address)

Arguments:
*address Pointer to destination page virtual address to erase.

Return Value: '0' if operation completed successfully

Remarks: None

Code Example:
NVMErasePage((void*) 0xBD000000);

NVMWriteRow

Description: This function programs a single row of program flash.

Include: plib.h

Prototype: unsigned int NVMWriteRow(void* address, void* data)

Arguments:
*address Pointer to destination row virtual address program.
*data Pointer to source data to write.

Return Value: '0' if operation completed successfully

Remarks: None

Code Example:
NVMWriteRow((void*) 0xBD000000, (void*) 0xA0000000);

NVMWriteWord

Description: This function programs a single word of program flash.

Include: plib.h
### NVMWriteWord (Continued)

<table>
<thead>
<tr>
<th>Prototype:</th>
<th>unsigned int NVMWriteWord(void* address, unsigned int data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arguments:</td>
<td>*address Pointer to destination word virtual address program. data Source data to write.</td>
</tr>
<tr>
<td>Return Value</td>
<td>'0' if operation completed successfully</td>
</tr>
<tr>
<td>Remarks:</td>
<td>None</td>
</tr>
<tr>
<td>Code Example:</td>
<td>NVMWriteWord((void*) 0xBD000000, 0x12345678);</td>
</tr>
</tbody>
</table>

### NVMClearError

<table>
<thead>
<tr>
<th>Description:</th>
<th>This function clears the error flag and resets the flash controller.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>plib.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td>unsigned int NVMClearError(void)</td>
</tr>
<tr>
<td>Arguments:</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>'0' if operation completed successfully</td>
</tr>
<tr>
<td>Remarks:</td>
<td>None</td>
</tr>
<tr>
<td>Code Example:</td>
<td>NVMClearError();</td>
</tr>
</tbody>
</table>

### NVMISError

<table>
<thead>
<tr>
<th>Description:</th>
<th>This function checks the error flags and return there value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>plib.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td>NVMISError()</td>
</tr>
<tr>
<td>Arguments:</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>'0' if error flag is not set</td>
</tr>
<tr>
<td>Remarks:</td>
<td>None</td>
</tr>
<tr>
<td>Code Example:</td>
<td>if(NVMISError()) NVMClearError();</td>
</tr>
</tbody>
</table>
6.0  RESET FUNCTIONS

The PIC32MX Reset library consists of functions and macros supporting common control features of this peripheral.

• Get Status Flag Operations
  mGetPORFlag
  mGetBORFlag
  mGetMCLRFlag
  mGetCMRFlag
  mGetWDTRFlag
  mGetSWRFlag
  mGetSleepFlag
  mGetIdleFlag
  mGetVregFlag

• Clear/Set Status Flag Operations
  mClearPORFlag
  mClearBORFlag
  mClearMCLRFlag
  mClearCMRFlag
  mClearWDTRFlag
  mClearSWRFlag
  mClearSleepFlag
  mClearIdleFlag
  mClearVregFlag
  mSetVregFlag

• PIC30F, PIC24H and PIC33F compatible operations
  PORStatReset
  BORStatReset
  isMCLR
  isPOR
  isBOR
  isWU
  isWDTTO
  isWDTWU
6.1 Get Status Flag Macros

### mGetPORFlag

**Description:** This function checks if Reset is due to Power-on Reset.

**Include:** plib.h

**Prototype:**

```c
unsigned int mGetPORFlag(void);
```

**Arguments:** None

**Return Value:** This function returns the RCON<POR> bit.
- If return value is '0x01', then reset is due to Power-on.
- If return value is '0', then no Power-on Reset occurred.

**Remarks:** None

**Source File:**

**Code Example:**

```c
unsigned int reset_state;
reset_state = mGetPORFlag();
```

### mGetBORFlag

**Description:** This function checks if Reset is due to Brown-out Reset.

**Include:** plib.h

**Prototype:**

```c
unsigned int isBOR(void);
```

**Arguments:** None

**Return Value:** This function returns the RCON<BOR> bit.
- If return value is not '0', then reset is due to brown-out.
- If return value is '0', then no brown-out occurred.

**Remarks:** None

**Source File:**

**Code Example:**

```c
unsigned int reset_state;
reset_state = mGetBORFlag();
```

### mGetMCLRFlag

**Description:** This function checks if Reset condition is due to MCLR pin going low.

**Include:** plib.h

**Prototype:**

```c
unsigned int mGetMCLRFlag(void);
```

**Arguments:** None

**Return Value:** This function returns the RCON<EXTR> bit.
- If return value is not '0x40', then Reset occurred due to MCLR pin going low.
- If return value is '0', then Reset is not due to MCLR going low.

**Remarks:** None

**Source File:**

**Code Example:**

```c
unsigned int reset_state;
reset_state = mGetMCLRFlag();
```
**mGetSWRFlag**

**Description:** This function checks if Reset is due to Software Reset.

**Include:** plib.h

**Prototype:**

```c
unsigned int mGetSWRFlag(void);
```

**Arguments:** None

**Return Value:** This function returns the RCON<SWR> bit.

- If return value is '0x20', then reset is due to Software Reset.
- If return value is '0', then no Software Reset occurred.

**Remarks:** None

**Source File:**

**Code Example:**

```c
unsigned int reset_state;
reset_state = mGetSWRFlag();
```

---

**mGetWDTOFlag**

**Description:** This function checks if Reset condition is due to WDT time-out.

**Include:** plib.h

**Prototype:**

```c
unsigned int mGetWDTOFlag(void);
```

**Arguments:** None

**Return Value:** This function returns the RCON<WDTO> bit.

- If return value is '0x10', then reset occurred due to WDT time-out.
- If return value is '0', then reset is not due to WDT time-out.

**Remarks:** None

**Source File:**

**Code Example:**

```c
unsigned int reset_state;
reset_state = mGetWDTOFlag();
```

---

**mGetCMRFlag**

**Description:** This function checks if Reset is due to Configuration Mis-match Reset.

**Include:** plib.h

**Prototype:**

```c
unsigned int mGetCMRFlag(void);
```

**Arguments:** None

**Return Value:** This function returns the RCON<CM> bit.

- If return value is '0x200', then reset is due to configuration mis-match.
- If return value is '0', then no configuration mis-match occurred.

**Remarks:** None

**Source File:**

**Code Example:**

```c
unsigned int reset_state;
reset_state = mGetCMRFlag();
```
mGetSLEEPFlag
Description: This function checks if the CPU was in SLEEP mode.
Include: plib.h
Prototype: unsigned int mGetSLEEPFlag(void);
Arguments: None
Return Value: This function returns the RCON<SLEEP> bit.
If return value is '0x08', then CPU was in SLEEP mode.
If return value is '0', then CPU was not in SLEEP mode.
Remarks: None
Source File: Code Example:
unsigned int sleep_state;
sleep_state = mGetSLEEPFlag();

mGetIDLEFlag
Description: This function checks if the CPU was in IDLE mode
Include: plib.h
Prototype: unsigned int mGetIDLEFlag(void);
Arguments: None
Return Value: This function returns the RCON<IDLE> bit.
If return value is '0x04', then CPU was in IDLE mode.
If return value is '0', then CPU was not in IDLE mode.
Remarks: None
Source File: Code Example:
unsigned int reset_state;
reset_state = mGetIDLEFlag();

mGetVREGSFlag
Description: This function checks the state of the VREG status flag.
Include: plib.h
Prototype: unsigned int mGetVREGSFlag(void);
Arguments: None
Return Value: This function returns the RCON<VREGS> bit.
If return value is '0x100', then VREG is enabled.
If return value is '0', then VREG is disabled.
Remarks: None
Source File: Code Example:
unsigned int reset_state;
reset_state = mGetVREGSFlag();
6.2 Clear/Set Status Flag Macros

mClearPORFlag
Description: This function clears POR (power on reset) status flag bit.
Include: plib.h
Prototype: mClearPORFlag(void);
Arguments: None
Return Value: This function clears the RCON<POR> bit.
Remarks: None
Source File: Code Example: mClearPORFlag();

mClearBORFlag
Description: This function clears BOR (brown out reset) status flag bit.
Include: plib.h
Prototype: mClearBORFlag(void);
Arguments: None
Return Value: This function clears the RCON<BOR> bit.
Remarks: None
Source File: Code Example: mClearBORFlag();

mClearMCLRFlag
Description: This function clears MCLR (master clear) status flag bit.
Include: plib.h
Prototype: mClearMCLRFlag(void);
Arguments: None
Return Value: This function clears the RCON<MCLR> bit.
Remarks: None
Source File: Code Example: mClearMCLRFlag();
### mClearSWRFlag

**Description:** This function clears SWR (software reset) status flag bit.

**Include:** plib.h

**Prototype:**

```c
mClearSWRFlag(void);
```

**Arguments:** None

**Return Value:** This function clears the RCON<SWR> bit.

**Remarks:** None

**Source File:**

**Code Example:**

```c
mClearSWRFlag();
```

### mClearWDTOFlag

**Description:** This function clears WDTO (watch dog timeout) status flag bit.

**Include:** plib.h

**Prototype:**

```c
mClearWDTOFlag(void);
```

**Arguments:** None

**Return Value:** This function clears the RCON<WDTO> bit.

**Remarks:** None

**Source File:**

**Code Example:**

```c
mClearWDTOFlag();
```

### mClearCMRFlag

**Description:** This function clears CM (configuration bits mismatch) status flag bit.

**Include:** plib.h

**Prototype:**

```c
mClearCMRFlag(void);
```

**Arguments:** None

**Return Value:** This function clears the RCON<CMR> bit.

**Remarks:** None

**Source File:**

**Code Example:**

```c
mClearCMRFlag();
```

### mClearSLEEPFlag

**Description:** This function clears SLEEP status flag bit.

**Include:** plib.h

**Prototype:**

```c
mClearSSLEEPFlag(void);
```

**Arguments:** None

**Return Value:** This function clears the RCON<SLEEP> bit.

**Remarks:** None

**Source File:**

**Code Example:**

```c
mClearSLEEPFlag();
```
mClearIDLEFlag

Description: This function clears IDLE status flag bit.
Include: plib.h
Prototype: mClearIdleFlag(void);
Arguments: None
Return Value: This function clears the RCON<IDLE> bit.
Remarks: None
Source File:
Code Example: mClearIDLEFlag();

mClearVREGSFlag

Description: This function disables the VREG.
Include: plib.h
Prototype: mClearVREGSFlag(void);
Arguments: None
Return Value: This function clears the RCON<VREGS> bit.
Remarks: None
Source File:
Code Example: mClearVREGSFlag();

mSetVREGSFlag

Description: This function enables the VREG.
Include: plib.h
Prototype: mSetVREGSFlag(void);
Arguments: None
Return Value: This function sets the RCON<VREGS> bit.
Remarks: None
Source File:
Code Example: mSetVREGSFlag();
6.3 PIC30F, PIC24H and PIC33F compatible macros

### isWDTTO

**Description:** This function checks if Reset condition is due to WDT time-out.

**Include:**       plib.h

**Prototype:**    unsigned int isWDTTO(void);

**Arguments:**    None

**Return Value:** This function returns the RCON:<WDTO> bit.
If return value is '0x10', then reset occurred due to WDT time-out.
If return value is '0', then reset is not due to WDT time-out.

**Remarks:**    None

**Source File:**

**Code Example:**

```c
unsigned int reset_state;
reset_state = isWDTTO();
```

### isWDTWU

**Description:** This function checks if Wake-up from SLEEP is due to WDT time-out.

**Include:**       plib.h

**Prototype:**    unsigned int isWDTWU(void);

**Arguments:**    None

**Return Value:** This function returns the status of RCON:<WDTO> and
RCON:<SLEEP> bits
If return value is '0x18', then Wake-up from SLEEP occurred due to
WDT time-out.
If return value is '0', then Wake-up from SLEEP is not due to WDT time-
out.

**Remarks:**    None

**Source File:**

**Code Example:**

```c
unsigned int reset_state;
reset_state = isWDTWU();
```
isWU

Description: This function checks if Wake-up from Sleep is due to MCLR, POR, BOR or Interrupt
Include: plib.h
Prototype: char isWU(void);
Arguments: None
Return Value: This function checks if Wake-up from Sleep has occurred. If yes, it checks for the cause for wake-up.
   if '0x01', wake-up is due to the occurrence of interrupt.
   if '0x02', wake-up is due to MCLR.
   if '0x04', wake-up is due to BOR.
   If Wake-up from Sleep has not occurred, then a value of '0' is returned.
Remarks: None
Source File: reset_is_wu.c
Code Example: char reset_state;
             reset_state = isWU();

PORStatReset

Description: This macro clears POR bit of RCON register.
Include: plib.h
Arguments: None
Remarks: None
Code Example: PORStatReset;

BORStatReset

Description: This macro clears BOR bit of RCON register.
Include: plib.h
Arguments: None
Remarks: None
Code Example: BORStatReset;
7.0 INTERRUPT FUNCTIONS

7.1 System Functions

**INTEnableSystemMultiVectoredInt**

Description: This function enables system wide multi-vector interrupt handling.
Include: plib.h
Prototype: void INTEnableSystemMultiVectoredInt(void)
Arguments: none.
Return Value: None
Remarks: User must call this function before any interrupts will be handled. The interrupts will go to the assigned vector location.

Source File: Code Example:
```
INTEnableSystemMultiVectoredInt();
```

**INTEnableSystemSingleVectoredInt**

Description: This function enables system wide single vectored interrupt handling.
Include: plib.h
Prototype: void INTEnableSystemSingleVectoredInt(void)
Arguments: none.
Return Value: None
Remarks: User must call this function before any interrupts will be handled. The interrupts will go to a single vector location.

Source File: Code Example:
```
INTEnableSystemSingleVectoredInt();
```

**INTDisableInterrupts**

Description: This function disables system wide interrupts.
Include: plib.h
Prototype: unsigned int INTDisableInterrupts(void)
Arguments: none.
Return Value: The previous state of the CP0 status register
Remarks: Disables all interrupts.

Source File: Code Example:
```
unsigned int status;
status = INTDisableInterrupts();
// .. do something with interrupts disabled
```

**INTEnableInterrupts**

Description: This function enables the microcontroller to receive system wide interrupts.
Include: plib.h
Prototype: void INTEnableInterrupts(void)
### INTEnableInterrupts

**Arguments:** none.

**Return Value**

The previous state of the CP0 status register

**Remarks:** Enables the microcontroller to handle interrupts.

**Source File:**

**Code Example:**

```c
unsigned int status;

status = INTEnableInterrupts();

// .. do something with interrupts enabled
```

### INTRestoreInterrupts

**Description:** This function restores the microcontroller to the passed state.

**Include:** plib.h

**Prototype:**

```c
void INTRestoreInterrupts(unsigned int status)
```

**Arguments:**

status - the status of the interrupts

0 - system wide interrupts are disabled
1 - system wide interrupts are enabled.

**Return Value**

none

**Remarks:** Restores the microcontroller’s handling of interrupts to the passed state

**Source File:**

**Code Example:**

```c
unsigned int status;

status = INTEnableInterrupts();

// .. do something with interrupts enabled

INTRestoreInterrupts(status);
```

### INTGetPendingInterrupt

**Description:** This function gets the pending interrupt.

**Include:** plib.h

**Prototype:**

```c
unsigned int INTGetPendingInterrupt(void)
```

**Arguments:**

none.

**Return Value**

The interrupt flag offset.

**Remarks:** The function will return the interrupt based on the natural priority. For example, the core timer will be serviced before the UART 1 receiver interrupt.

**Source File:**

**Code Example:**

```c
unsigned int int_num;

while(int_num = INTGetPendingInterrupt())
{
    // service interrupt
}
```
### INTClearFlag

**Description:** This function clears the interrupt flag.

**Include:** plib.h

**Prototype:**
```c
void INTClearFlag(INT_SOURCE source)
```

**Arguments:**
- `source` - the interrupt to be cleared

**Return Value:** none

**Remarks:**
This function will clear the interrupt flag of the passed value "source". See IRQ table for more information

**Source File:**

**Code Example:**
```c
// clear the core timer interrupt
INTClearFlag(INT_CT);
```

### INTSetFlag

**Description:** This function sets the interrupt flag.

**Include:** plib.h

**Prototype:**
```c
void INTSetFlag(INT_SOURCE source)
```

**Arguments:**
- `source` - the interrupt to be cleared

**Return Value:** none

**Remarks:**
This function will set the interrupt flag of the passed value "source". See IRQ table for more information

**Source File:**

**Code Example:**
```c
// set the core timer interrupt
INTSetFlag(INT_CT);
```

### INTGetFlag

**Description:** This function gets the interrupt flag.

**Include:** plib.h

**Prototype:**
```c
unsigned int INTGetFlag(INT_SOURCE source)
```

**Arguments:**
- `source` - the interrupt to be cleared

**Return Value:**
- the value of the interrupt flag

**Remarks:**
This function will get the interrupt flag of the passed value "source". See IRQ table for more information

**Source File:**

**Code Example:**
```c
// get the core timer interrupt flag
unsigned int flag;
flag = INTGetFlag(INT_CT);
```

### INTEnable

**Description:** This function enables or disables an interrupt.

**Include:** plib.h
### INTEnable

**Prototype:**
```
void INTEnable(INT_SOURCE source, unsigned int enable)
```

**Arguments:**
- `source`: the interrupt to be cleared
- `enable`: 0 to disable, 1 to enable interrupt

**Return Value:**
none

**Remarks:**
This function will enable or disable the interrupt of the passed value `source`. See IRQ table for more information

**Source File:**
plib.h

**Code Example:**
```
// enable the core timer interrupt
INTEnable(INT_CT, 1);

// disable the core timer interrupt
INTEnable(INT_CT, 0);
```

### INTGetEnable

**Description:**
This function get the enable/disable status of the interrupt.

**Include:**
plib.h

**Prototype:**
```
unsigned int INTEnable(INT_SOURCE source)
```

**Arguments:**
- `source`: the interrupt to be cleared

**Return Value:**
0 if disabled, else enabled

**Remarks:**
This function will provide the enable/disables status of the interrupt of the passed value `source`. See IRQ table for more information

**Source File:**
plib.h

**Code Example:**
```
// get the enable/disable status core timer interrupt
unsigned int enable;
enable = INTGetEnable(INT_CT);
```

### INTSetPriority

**Description:**
This function sets the interrupt priority.

**Include:**
plib.h

**Prototype:**
```
void INTSetPriority(INT_SOURCE source, unsigned int priority)
```

**Arguments:**
- `source`: the interrupt to be cleared
- `priority`: value 1 - 7

**Return Value:**
none

**Remarks:**
This function will set the interrupt priority of interrupt the passed value `source`. See IRQ table for more information

**Source File:**
plib.h

**Code Example:**
```
// set core timer interrupt priority two
INTSetPriority(INT_CT, INT_PRIORITY_LEVEL_2);
```

### INTGetPriority

**Description:**
This function gets the interrupt priority.

**Include:**
plib.h
INTGetPriority

Prototype: unsigned int INTGetPriority(INT_SOURCE source)
Arguments: source - the interrupt to be cleared
Return Value: the current priority (0 - 7)
Remarks: This function will get the interrupt priority of interrupt the passed value "source". See IRQ table for more information

Source File:
Code Example: // get core timer interrupt
unsigned int priority;

priority = INTGetPriority(INT_CT);

INTSetSubPriority

Description: This function sets the sub-interrupt priority.
Include: plib.h
Prototype: void INTSetPriority(INT_SOURCE source, unsigned int subPriority)
Arguments: source - the interrupt to be cleared subPriority - value 0 - 3
Return Value: none
Remarks: This function will set the interrupt sub-priority of interrupt the passed value "source". See IRQ table for more information

Source File:
Code Example: // set core timer sub-interrupt priority one
INTSetPriority(INT_CT, INT_SUB_PRIORITY_LEVEL_1);

INTGetSubPriority

Description: This function gets the sub-interrupt priority.
Include: plib.h
Prototype: unsigned int INTGetSubPriority(INT_SOURCE source)
Arguments: source - the interrupt whose sub-priority is to be returned
Return Value: the current sub-priority (0 - 3)
Remarks: This function will get the sub-interrupt priority of interrupt the passed value "source". See IRQ table for more information

Source File:
Code Example: // get core timer sub-interrupt
unsigned int sub_priority;

sub_priority = INTGetSubPriority(INT_CT);

TABLE 8-1: INTERRUPT ENUMERATIONS

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT_CT</td>
<td>Core Timer Interrupt</td>
</tr>
<tr>
<td>INT_CS0</td>
<td>Core Software Interrupt 0</td>
</tr>
<tr>
<td>Enumeration</td>
<td>Peripheral</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>INT_CS1</td>
<td>Core Software Interrupt 1</td>
</tr>
<tr>
<td>INT_INT0</td>
<td>External Interrupt 0</td>
</tr>
<tr>
<td>INT_T1</td>
<td>Timer 1 Interrupt</td>
</tr>
<tr>
<td>INT_IC1</td>
<td>Input Capture 1 Interrupt</td>
</tr>
<tr>
<td>INT_OC1</td>
<td>Output Compare 1 Interrupt</td>
</tr>
<tr>
<td>INT_INT1</td>
<td>External Interrupt 1</td>
</tr>
<tr>
<td>INT_T2</td>
<td>Timer 2 Interrupt</td>
</tr>
<tr>
<td>INT_IC2</td>
<td>Input Capture 2 Interrupt</td>
</tr>
<tr>
<td>INT_OC2</td>
<td>Output Compare 2 Interrupt</td>
</tr>
<tr>
<td>INT_INT2</td>
<td>External Interrupt 2</td>
</tr>
<tr>
<td>INT_T3</td>
<td>Timer 3 Interrupt</td>
</tr>
<tr>
<td>INT_IC3</td>
<td>Input Capture 3 Interrupt</td>
</tr>
<tr>
<td>INT_OC3</td>
<td>Output Compare 3 Interrupt</td>
</tr>
<tr>
<td>INT_INT3</td>
<td>External Interrupt 3</td>
</tr>
<tr>
<td>INT_T4</td>
<td>Timer 4 Interrupt</td>
</tr>
<tr>
<td>INT_IC4</td>
<td>Input Capture 4 Interrupt</td>
</tr>
<tr>
<td>INT_OC4</td>
<td>Output Compare 4 Interrupt</td>
</tr>
<tr>
<td>INT_INT4</td>
<td>External Interrupt 4</td>
</tr>
<tr>
<td>INT_T5</td>
<td>Timer 5 Interrupt</td>
</tr>
<tr>
<td>INT_IC5</td>
<td>Input Capture 5 Interrupt</td>
</tr>
<tr>
<td>INT_OC5</td>
<td>Output Compare 5 Interrupt</td>
</tr>
<tr>
<td>INT_CN</td>
<td>Input Change Interrupt</td>
</tr>
<tr>
<td>INT_SPI1E</td>
<td>SPI 1 Fault</td>
</tr>
<tr>
<td>INT_SPI1TX</td>
<td>SPI 1 Transfer Done</td>
</tr>
<tr>
<td>INT_SPI1RX</td>
<td>SPI 1 Receiver Done</td>
</tr>
<tr>
<td>INT_SPI1</td>
<td>SPI 1</td>
</tr>
<tr>
<td>INT_U1E</td>
<td>UART 1 Error</td>
</tr>
<tr>
<td>INT_U1RX</td>
<td>UART 1 Receiver</td>
</tr>
<tr>
<td>INT_U1TX</td>
<td>UART 1 Transmitter</td>
</tr>
<tr>
<td>INT_U1</td>
<td>UART 1</td>
</tr>
<tr>
<td>INT_I2C1B</td>
<td>I2C 1 Bus Collision Event</td>
</tr>
<tr>
<td>INT_I2C1S</td>
<td>I2C 1 Slave Event</td>
</tr>
<tr>
<td>INT_I2C1M</td>
<td>I2C 1 Master Event</td>
</tr>
<tr>
<td>INT_I2C1</td>
<td>I2C 1</td>
</tr>
<tr>
<td>INT_AD1</td>
<td>ADC Convert Done</td>
</tr>
<tr>
<td>INT_PMP</td>
<td>Parallel Master Port Interrupt</td>
</tr>
<tr>
<td>INT_CMP1</td>
<td>Comparator 1 Interrupt</td>
</tr>
<tr>
<td>INT_CMP2</td>
<td>Comparator 2 Interrupt</td>
</tr>
<tr>
<td>INT_SPI2E</td>
<td>SPI 2 Fault</td>
</tr>
<tr>
<td>INT_SPI2TX</td>
<td>SPI 2 Transfer Done</td>
</tr>
<tr>
<td>INT_SPI2RX</td>
<td>SPI 2 Receiver Done</td>
</tr>
<tr>
<td>INT_SPI2</td>
<td>SPI 2</td>
</tr>
<tr>
<td>INT_U2E</td>
<td>UART 2 Error</td>
</tr>
<tr>
<td>INT_U2RX</td>
<td>UART 2 Receiver</td>
</tr>
<tr>
<td>INT_U2TX</td>
<td>UART 2 Transmitter</td>
</tr>
</tbody>
</table>
### 7.2 Inline Functions

#### INTGetInterruptVectorNumberAndPriority

**Description:**
This function gets the pending vector number and its priority.

**Include:**
plib.h

**Prototype:**
```c
extern inline void __attribute__ ((always_inline))
INTGetInterruptVectorNumberAndPriority(unsigned int*
number, unsigned int *priority)
```

**Arguments:**
- `number` - a pointer to where the vector number will be stored
- `priority` - a pointer to where the vector number’s priority will be stored

**Return Value:**
None

**Remarks:**
None.

**Source File:**
None

**Code Example:**
```c
unsigned int vector, priority;
INTGetInterruptVectorNumberAndPriority(&vector,
&priority);
```

### 7.3 System Macros

#### mClearIFSRegister

**Description:**
This macro clears the Interrupt Flag register.

**Include:**
plib.h

**Prototype:**
```c
void mClearIFSRegister(reg_num)
```
### mClearIFSRegister

**Arguments:**
- `reg_num` - the IFS index to clear (reg_num = 0 would mean that IFS0 would be cleared).

**Return Value:** None

**Remarks:** None.

**Source File:** None

**Code Example:**

```c
mClearIFSRegister(0);
```

### mClearIECRegister

**Description:**
This macro clears the Interrupt Enable register.

**Include:**
plib.h

**Prototype:**
```c
void mClearIECRegister(reg_num);
```

**Arguments:**
- `reg_num` - the IEC index to clear (reg_num = 0 would mean that IEC0 would be cleared).

**Return Value:** None

**Remarks:** Set the edge that the external interrupt will generate an interrupt.

**Source File:** None

**Code Example:**

```c
mClearIECRegister(0);
```

### mClearAllIFSRegister

**Description:**
This macro clears all the bits in all of the IFS registers

**Include:**
plib.h

**Prototype:**
```c
void mClearAllIFSRegister(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** None.

**Source File:** None

**Code Example:**

```c
mClearAllIFSRegister();
```

### mClearAllIECRegister

**Description:**
This macro clears all the bits in all of the IEC registers

**Include:**
plib.h

**Prototype:**
```c
void mClearAllIECRegister(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** None.

**Source File:** None

**Code Example:**

```c
mClearAllIECRegister();
```
mINTSetIFSx

**Description:** This macro sets bits in the IFSx register

**Include:** plib.h

**Prototype:** void mINTSetIFSx(unsigned int flag)

**Arguments:** flag - bits to set

**Return Value** None

**Remarks:** The macro is for all IFS registers. If one would like to set bits in the IFS1 register, they need to replace the ‘x’ with 1

**Source File:** None

**Code Example:**

```c
mINTSetIFS0(1);
mINTSetIFS1(2);
mINTSetIFS2(4);
```

mINTClearIFSx

**Description:** This macro clears bits in the IFSx register

**Include:** plib.h

**Prototype:** void mINTClearIFSx(unsigned int flag)

**Arguments:** flag - bits to clear

**Return Value** None

**Remarks:** The macro is for all IFS registers. If one would like to clear bits in the IFS1 register, they need to replace the ‘x’ with 1

**Source File:** None

**Code Example:**

```c
mINTClearIFS0(1);
mINTClearIFS1(2);
mINTClearIFS2(4);
```

mINTGetIFSx

**Description:** This macro gets bits in the IFSx register

**Include:** plib.h

**Prototype:** unsigned int mINTGetIFSx(unsigned int flag)

**Arguments:** flag - bits to get

**Return Value** None

**Remarks:** The macro is for all IFS registers. If one would like to clear bits in the IFS1 register, they need to replace the ‘x’ with 1

**Source File:** None

**Code Example:**

```c
if(!mINTGetIFS0(1))
    return;
if(mINTGetIFS1(2) == 2)
    return;
if(mINTGetIFS2(3) != 3)
    return;
```
### mINTSetIECx

**Description:** This macro sets bits in the IECx register  
**Include:** plib.h  
**Prototype:** void mINTSetIECx(unsigned int flag)  
**Arguments:** flag - bits to set  
**Return Value:** None  
**Remarks:** The macro is for all IEC registers. If one would like to set bits in the IEC1 register, they need to replace the ‘x’ with 1  
**Source File:** None  
**Code Example:**
```
  mINTSetIEC0(1);
  mINTSetIEC1(2);
  mINTSetIEC2(4);
```

### mINTClearIECx

**Description:** This macro clears bits in the IECx register  
**Include:** plib.h  
**Prototype:** void mINTClearIECx(unsigned int flag)  
**Arguments:** flag - bits to clear  
**Return Value:** None  
**Remarks:** The macro is for all IEC registers. If one would like to clear bits in the IEC1 register, they need to replace the ‘x’ with 1  
**Source File:** None  
**Code Example:**
```
  mINTClearIEC0(1);
  mINTClearIEC1(2);
  mINTClearIEC2(4);
```

### mINTSetIntProximityTimerReload

**Description:** This macro sets the 16 bit proximity timer  
**Include:** plib.h  
**Prototype:** void mINTSetIntProximityTimerReload(unsigned int time)  
**Arguments:** time - 32 bit value that will be loaded into the proximity timer  
**Return Value:** None  
**Remarks:** None  
**Source File:** None.c  
**Code Example:**
```
  mINTSetIntProximityTimerReload(0x0080000);
```

### mINTGetIntProximityTimer

**Description:** This macro gets the current value of the proximity timer.
mINTGetIntProximityTimer

Include: plib.h
Prototype: unsigned int mINTGetIntProximityTimer(void)
Arguments: None
Return Value: The current value of the proximity timer.
Remarks: If the proximity timer has not been triggered, the value that will be read back is the reload time.
Source File: None
Code Example:

```c
unsigned short time;

  time = mINTGetIntProximityTimer();
  if(time < 4000)
  ....
```

mINTSetFreeze

Description: This macro sets the freeze bit.
Include: plib.h
Prototype: void mINTSetFreeze(void)
Arguments: None
Return Value: None
Remarks: The device must be in debug mode.
Source File: None
Code Example: mINTSetFreeze();

mINTClearFreeze

Description: This macro clears the freeze bit.
Include: plib.h
Prototype: void mINTClearFreeze(void)
Arguments: None
Return Value: None
Remarks: The device must be in debug mode.
Source File: None
Code Example: mINTClearFreeze();

mINTSetTemporalProximityControl

Description: This macro sets the temporary proximity control level.
Include: plib.h
Prototype: void mINTSetTemporalProximityControl(unsigned int level)
**mINTSetTemporalProximityControl**

**Arguments:**
- level - the interrupt level for the proximity timer to trigger on
  - 0 - timer disabled
  - 1 - timer triggered for level 1 interrupts
  - 2 - timer triggered for level 2 interrupts or lower
  - 3 - timer triggered for level 3 interrupts or lower
  - 4 - timer triggered for level 4 interrupts or lower
  - 5 - timer triggered for level 5 interrupts or lower
  - 6 - timer triggered for level 6 interrupts or lower
  - 7 - timer triggered for level 7 interrupts or lower

**Return Value:** None

**Remarks:** None.

**Source File:** None

**Code Example:**
```
mINTSetTemporalProximityControl(2);
```

---

**mINTDisableTemporalProximityControl**

**Description:** This macro disables the temporal proximity timer.

**Include:** plib.h

**Prototype:**
```
void mINTDisableTemporalProximityControl(void)
```

**Arguments:** None

**Return Value:** None

**Remarks:** None.

**Source File:** None

**Code Example:**
```
mINTSetTemporalProximityControl(2);
...
mINTDisableTemporalProximityControl();
```

---

**mINTSingleVectorRegisterSelect**

**Description:** This selects the general purpose register set that will be used by the single vector handler.

**Include:** plib.h

**Prototype:**
```
void mINTSingleVectorRegisterSelect(unsigned int reg)
```

**Arguments:**
- reg - the register set that will be used
  - 0 - the general register set that is used for all CPU functions
  - 1 - the shadow register set

**Return Value:** None

**Remarks:** None.

**Source File:** None

**Code Example:**
```
mINTSingleVectorRegisterSelect(0);
```
### mINTGetInterruptVectorNumber

**Description:** This macro will get the highest pending priority interrupt vector

**Include:** plib.h

**Prototype:**

```c
unsigned int mINTGetInterruptVectorNumber(void)
```

**Arguments:** None

**Return Value**

The highest pending interrupt vector

**Remarks:** None.

**Source File:** None

**Code Example:**

```c
unsigned int vector;

vector = mINTGetInterruptVectorNumber();
```

### mINTGetInterruptVectorPriority

**Description:** This macro will get the highest pending priority

**Include:** plib.h

**Prototype:**

```c
unsigned int mINTGetInterruptVectorPriority(void)
```

**Arguments:** None

**Return Value**

The highest pending interrupt priority.

**Remarks:** If all of the pending interrupts have been processed, this macro will return 0.

**Source File:** None

**Code Example:**

```c
unsigned int priority;

priority = mINTGetInterruptVectorPriority();
```

### mINTDisableSystemMultiVectorInt

**Description:** This macro will disable system wide multi-vectored interrupts

**Include:** plib.h

**Prototype:**

```c
void mINTDisableSystemMultiVectoredInt(void)
```

**Arguments:** None

**Return Value**

None.

**Remarks:** Will disable multi-vectored interrupts.

**Source File:** None

**Code Example:**

```c
mINTDisableSystemMultiVectoredInt();
```

### mINTDisableSystemSingleVectorInt

**Description:** This macro will disable system wide single vectored interrupts

**Include:** plib.h

**Prototype:**

```c
void mINTDisableSystemSingleVectoredInt(void)
```

**Arguments:** None
### 7.4 Peripheral Interrupt Macros

#### 7.4.1 PERIPHERAL INTERRUPT EVENT MACROS

<table>
<thead>
<tr>
<th>Description</th>
<th>This clears the peripheral interrupt flag.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>plib.h</td>
</tr>
<tr>
<td>Prototype</td>
<td>void m(xx)ClearIntFlag(void)</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
<tr>
<td>Remarks</td>
<td>Replace (xx) with the corresponding peripheral from the macro flag table.</td>
</tr>
<tr>
<td>Source File</td>
<td>None</td>
</tr>
<tr>
<td>Code Example</td>
<td>// clearing the Interrupt Flag for the Core Timer mCTClearIntFlag();</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>This gets the peripheral interrupt flag.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>plib.h</td>
</tr>
<tr>
<td>Prototype</td>
<td>void m(xx)GetIntFlag(void)</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
<tr>
<td>Remarks</td>
<td>Replace (xx) with the corresponding peripheral from the macro flag table.</td>
</tr>
<tr>
<td>Source File</td>
<td>None</td>
</tr>
<tr>
<td>Code Example</td>
<td>// gets the Interrupt Flag for the Core Timer mCTGetIntFlag();</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>This sets or clears the interrupt enable for the specific peripheral.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>plib.h</td>
</tr>
<tr>
<td>Prototype</td>
<td>void m(xx)IntFlag(unsigned int enable)</td>
</tr>
<tr>
<td>Arguments</td>
<td>enable</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>
| Remarks     | 0 - disable the peripheral interrupt  
               1 - enable the peripheral interrupt                          |
| Source File | None                                                                |
| Code Example| // sets the Interrupt enable for the Core Timer mCTIntEnable();     |
**m(xx)IntEnable**

**Remarks:** Replace (xx) with the corresponding peripheral from the macro flag table.

**Source File:** None

**Code Example:**
```c
// sets the interrupt enable for the Core Timer
mCTIntEnable(1);
```

### TABLE 8-2: PERIPHERAL FLAGS TO MACRO ABBREVIATIONS

<table>
<thead>
<tr>
<th>Macro Abreviation(xx)</th>
<th>Peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT</td>
<td>Core Timer Interrupt</td>
</tr>
<tr>
<td>CS0</td>
<td>Core Software Interrupt 0</td>
</tr>
<tr>
<td>CS1</td>
<td>Core Software Interrupt 1</td>
</tr>
<tr>
<td>INT0</td>
<td>External Interrupt 0</td>
</tr>
<tr>
<td>T1</td>
<td>Timer 1 Interrupt</td>
</tr>
<tr>
<td>IC1</td>
<td>Input Capture 1 Interrupt</td>
</tr>
<tr>
<td>OC1</td>
<td>Output Compare 1 Interrupt</td>
</tr>
<tr>
<td>INT1</td>
<td>External Interrupt 1</td>
</tr>
<tr>
<td>T2</td>
<td>Timer 2 Interrupt</td>
</tr>
<tr>
<td>IC2</td>
<td>Input Capture 2 Interrupt</td>
</tr>
<tr>
<td>OC2</td>
<td>Output Compare 2 Interrupt</td>
</tr>
<tr>
<td>INT2</td>
<td>External Interrupt 2</td>
</tr>
<tr>
<td>T3</td>
<td>Timer 3 Interrupt</td>
</tr>
<tr>
<td>IC3</td>
<td>Input Capture 3 Interrupt</td>
</tr>
<tr>
<td>OC3</td>
<td>Output Compare 3 Interrupt</td>
</tr>
<tr>
<td>INT3</td>
<td>External Interrupt 3</td>
</tr>
<tr>
<td>T4</td>
<td>Timer 4 Interrupt</td>
</tr>
<tr>
<td>IC4</td>
<td>Input Capture 4 Interrupt</td>
</tr>
<tr>
<td>OC4</td>
<td>Output Compare 4 Interrupt</td>
</tr>
<tr>
<td>INT4</td>
<td>External Interrupt 4</td>
</tr>
<tr>
<td>T5</td>
<td>Timer 5 Interrupt</td>
</tr>
<tr>
<td>IC5</td>
<td>Input Capture 5 Interrupt</td>
</tr>
<tr>
<td>OC5</td>
<td>Output Compare 5 Interrupt</td>
</tr>
<tr>
<td>CN</td>
<td>Input Change Interrupt</td>
</tr>
<tr>
<td>SPI1E</td>
<td>SPI 1 Fault</td>
</tr>
<tr>
<td>SPI1TX</td>
<td>SPI 1 Transfer Done</td>
</tr>
<tr>
<td>SPI1RX</td>
<td>SPI 1 Receiver Done</td>
</tr>
<tr>
<td>U1E</td>
<td>UART 1 Error</td>
</tr>
<tr>
<td>U1RX</td>
<td>UART 1 Receiver</td>
</tr>
<tr>
<td>U1TX</td>
<td>UART 1 Transmitter</td>
</tr>
<tr>
<td>I2C1B</td>
<td>I2C 1 Bus Collision Event</td>
</tr>
<tr>
<td>I2C1S</td>
<td>I2C 1 Slave Event</td>
</tr>
<tr>
<td>I2C1M</td>
<td>I2C 1 Master Event</td>
</tr>
<tr>
<td>AD1</td>
<td>ADC Convert Done</td>
</tr>
<tr>
<td>PMP</td>
<td>Parallel Master Port Interrupt</td>
</tr>
<tr>
<td>CMP1</td>
<td>Comparator 1 Interrupt</td>
</tr>
</tbody>
</table>
### 7.4.2 Peripheral Interrupt Vector Macros

**m(yy)SetIntPriority**

**Description:**
This macro set is peripheral interrupt vector priority.

**Include:**
plib.h

**Prototype:**
```c
void m(yy)SetIntPriority(unsigned int priority)
```

**Arguments:**
```
priority
0 - disable interrupt
1 - priority level 1
2 - priority level 2
3 - priority level 3
4 - priority level 4
5 - priority level 5
6 - priority level 6
7 - priority level 7
```

**Return Value:**
None

**Remarks:**
Replace (yy) with the corresponding peripheral from the macro interrupt vector table.

**Source File:**
None

**Code Example:**
```c
// sets the interrupt priority level for the Core Timer
mCTSetIntPriority(1);
```
m(yy)GetIntPriority

Description: This macro gets the current peripheral interrupt vector priority.
Include: plib.h
Prototype: unsigned int m(yy)GetIntPriority(void)
Arguments: None
Return Value:
- 0 - disable interrupt
- 1 - priority level 1
- 2 - priority level 2
- 3 - priority level 3
- 4 - priority level 4
- 5 - priority level 5
- 6 - priority level 6
- 7 - priority level 7

Remarks: Replace (yy) with the corresponding peripheral from the macro interrupt vector table.
Source File: None
Code Example:
```c
// sets the interrupt priority level for the Core Timer
unsigned int priority;
priority = mCTGetIntPriority();
```

m(yy)SetIntSubPriority

Description: This macro sets peripheral interrupt vector sub-priority.
Include: plib.h
Prototype: void m(yy)SetIntSubPriority(unsigned int subPriority)
Arguments: subPriority
- 0 - sub-priority level 0
- 1 - sub-priority level 1
- 2 - sub-priority level 2
- 3 - sub-priority level 3

Return Value: None
Remarks: Replace (yy) with the corresponding peripheral from the macro interrupt vector table.
Source File: None
Code Example:
```c
// sets the interrupt sub-priority level for the Core Timer
mCTSetIntSubPriority(1);
```

m(yy)GetIntSubPriority

Description: This macro gets the peripheral interrupt vector sub-priority.
Include: plib.h
m(yy)GetIntSubPriority

Prototype: unsigned int m(yy)GetIntSubPriority(void)
Arguments: None
Return Value: 0 - sub-priority level 0
1 - sub-priority level 1
2 - sub-priority level 2
3 - sub-priority level 3
Remarks: Replace (yy) with the corresponding peripheral from the macro interrupt vector table.
Source File: None
Code Example:

```c
// gets the interrupt sub-priority level for the Core Timer

unsigned int sub;
sub = mCTGetIntSubPriority();
```

TABLE 8-3: PHERIPHERAL VECTOR TO MACRO ABERIVATIONS

<table>
<thead>
<tr>
<th>Macro Abreviation(yy)</th>
<th>Peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT</td>
<td>Core Timer Vector</td>
</tr>
<tr>
<td>CS0</td>
<td>Core Software Vector 0</td>
</tr>
<tr>
<td>CS1</td>
<td>Core Software Vector 1</td>
</tr>
<tr>
<td>INT0</td>
<td>External Vector 0</td>
</tr>
<tr>
<td>T1</td>
<td>Timer 1 Vector</td>
</tr>
<tr>
<td>IC1</td>
<td>Input Capture 1 Vector</td>
</tr>
<tr>
<td>OC1</td>
<td>Output Compare 1 Vector</td>
</tr>
<tr>
<td>INT1</td>
<td>External Vector 1</td>
</tr>
<tr>
<td>T2</td>
<td>Timer 2 Vector</td>
</tr>
<tr>
<td>IC2</td>
<td>Input Capture 2 Vector</td>
</tr>
<tr>
<td>OC2</td>
<td>Output Compare 2 Vector</td>
</tr>
<tr>
<td>INT2</td>
<td>External Vector 2</td>
</tr>
<tr>
<td>T3</td>
<td>Timer 3 Vector</td>
</tr>
<tr>
<td>IC3</td>
<td>Input Capture 3 Vector</td>
</tr>
<tr>
<td>OC3</td>
<td>Output Compare 3 Vector</td>
</tr>
<tr>
<td>INT3</td>
<td>External Vector 3</td>
</tr>
<tr>
<td>T4</td>
<td>Timer 4 Vector</td>
</tr>
<tr>
<td>IC4</td>
<td>Input Capture 4 Vector</td>
</tr>
<tr>
<td>OC4</td>
<td>Output Compare 4 Vector</td>
</tr>
<tr>
<td>INT4</td>
<td>External Vector 4</td>
</tr>
<tr>
<td>T5</td>
<td>Timer 5 Vector</td>
</tr>
<tr>
<td>IC5</td>
<td>Input Capture 5 Vector</td>
</tr>
<tr>
<td>OC5</td>
<td>Output Compare 5 Vector</td>
</tr>
<tr>
<td>CN</td>
<td>Input Change Vector</td>
</tr>
<tr>
<td>SPI1</td>
<td>SPI 1 Vector</td>
</tr>
<tr>
<td>U1</td>
<td>UART 1 Vector</td>
</tr>
<tr>
<td>I2C1</td>
<td>I2C 1 Vector</td>
</tr>
<tr>
<td>AD1</td>
<td>ADC Convert Done Vector</td>
</tr>
</tbody>
</table>
7.4.3 PERIPHERAL INTERRUPT MULTI-EVENT MACROS

### m(zz)ClearAllIntFlag

**Description:**
This clears all of the interrupt flags associated with the peripheral interrupt.

**Include:**
plib.h

**Prototype:**
void m(zz)ClearAllIntFlag(void)

**Arguments:**
None

**Return Value:**
None

**Remarks:**
Replace (zz) with the corresponding peripheral from the macro flag table.

**Source File:**
None

**Code Example:**
```
// clearing all Interrupt Flags SPI 1 Peripheral
mSPI1ClearAllIntFlags();
```

### m(zz)IntDisable

**Description:**
This disables all of the interrupts associated with the peripheral.

**Include:**
plib.h

**Prototype:**
void m(zz)IntDisable(void)

**Arguments:**
None

**Return Value:**
None

**Remarks:**
Replace (zz) with the corresponding peripheral from the macro flag table.

**Source File:**
None
m(zz)IntDisable

Source File: None
Code Example: // disables all Interrupts SPI 1 Peripheral
mSPI1IntDisable();

TABLE 8-4: MULTI-EVENT PERIPHERAL TO MACROS ABERIVATION

<table>
<thead>
<tr>
<th>Marco Aberivation(zz)</th>
<th>Multi-Event Peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI1</td>
<td>SPI 1</td>
</tr>
<tr>
<td>U1</td>
<td>UART 1</td>
</tr>
<tr>
<td>I2C1</td>
<td>I2C 1</td>
</tr>
<tr>
<td>SPI2</td>
<td>SPI 2</td>
</tr>
<tr>
<td>U2</td>
<td>UART 2</td>
</tr>
<tr>
<td>I2C2</td>
<td>I2C 2</td>
</tr>
</tbody>
</table>

7.5 Software Interrupt

mConfigIntCoreSW0
mConfigIntCoreSW1

Description: Configures the priority, sub priority and enables the core software interrupt.

Include: plib.h

Prototype: void mConfigIntCoreSW0(config)
void mConfigIntCoreSW1(config)

Arguments: config Individual interrupt enable/disable information as defined below:

- **Interrupt enable**
  - CSW_INT_ON
  - CSW_INT_OFF

- **Interrupt Priority**
  - CSW_INT_INT_PR0
  - CSW_INT_INT_PR1
  - CSW_INT_INT_PR2
  - CSW_INT_INT_PR3
  - CSW_INT_INT_PR4
  - CSW_INT_INT_PR5
  - CSW_INT_INT_PR6
  - CSW_INT_INT_PR7

- **Interrupt Sub-Priority**
  - CSW_INT_SUB_PRIOR_0
  - CSW_INT_SUB_PRIOR_1
  - CSW_INT_SUB_PRIOR_2
  - CSW_INT_SUB_PRIOR_3

Return Value: None
Remarks: None
Source File: None
mConfigIntCoreSW0
mConfigIntCoreSW1

Code Example:  // set up the core software interrupt with a priority of 3 and zero sub-priority
              mConfigIntCoreSW0((CSW_INT_ON | CSW_INT_PRIOR_3 | CSW_INT_SUB_PRIOR_0));

mEnableIntCoreSW0
mEnableIntCoreSW1

Description:  This enables the core software interrupt.
Include:  plib.h
Prototype:  void mEnableIntCoreSW0(void)
            void mEnableIntCoreSW1(void)
Arguments:  None
Return Value  None
Remarks:  none
Source File:  None
Code Example:  // enable the core software interrupt
               mEnableIntCoreSW0();

mDisableIntCoreSW0
mDisableIntCoreSW1

Description:  This disables the core software interrupt.
Include:  plib.h
Prototype:  void mDisableIntCoreSW0(void)
            void mDisableIntCoreSW1(void)
Arguments:  None
Return Value  None
Remarks:  none
Source File:  None
Code Example:  // disable the core software interrupt
               mDisableIntCoreSW0();

mSetPriorityIntCoreSw0
mSetPriorityIntCoreSw1

Description:  This sets the priority of the software interrupt.
Include:  plib.h
Prototype:  void mSetPriorityIntCoreSW0(priority)
            void mSetPriorityIntCoreSW1(priority)
Arguments:  priority - the interrupt priority
### mSetPriorityIntCoreSw0

**Description:** This sets the core software interrupt.

**Include:** plib.h

**Prototype:**

```c
void mSetPriorityIntCoreSw0(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** This will generate a software interrupt.

**Source File:** None

**Code Example:**

```c
// set the core software interrupt to priority level 6
mSetPriorityIntCoreSw0(CSW_INT_INT_PR6);
```

### mSetPriorityIntCoreSw1

**Description:** This sets the core software interrupt.

**Include:** plib.h

**Prototype:**

```c
void mSetPriorityIntCoreSw1(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** The user must clear the software interrupt using this function and also the interrupt flag to clear the interrupt request.

**Source File:** None

**Code Example:**

```c
// set the core software interrupt to priority level 6
mSetPriorityIntCoreSw1(CSW_INT_INT_PR6);
```

### SetCoreSw0

**Description:** This sets the core software interrupt.

**Include:** plib.h

**Prototype:**

```c
void SetCoreSw0(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** This will generate a software interrupt.

**Source File:** None

**Code Example:**

```c
// generate a software interrupt
SetCoreSW0();
```

### ClearCoreSw0

**Description:** This sets the core software interrupt.

**Include:** plib.h

**Prototype:**

```c
void ClearCoreSw0(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** The user must clear the software interrupt using this function and also the interrupt flag to clear the interrupt request.

**Source File:** None

**Code Example:**

```c
// clear the software interrupt
ClearCoreSw0();
```
8.0 OSCILLATOR FUNCTIONS

The PIC32MX has multiple clock sources, with varying degrees of adjustability. The oscillator library functions are available to allow high-level control of the clock source and scaling of the frequency at runtime. The following functions and macros are available:

mOSCClockFailStatus() - Returns the status of the Clock Fail bit.

mOSCDisableSOSC() - Clears the secondary oscillator request. The secondary oscillator will be turned off if it is not being used by the CPU or a peripheral.

mOSCEnableSOSC() - Sets the secondary oscillator request. The secondary oscillator will be turned on.

mOSCGetPBDIV() - Returns the peripheral bus divisor value.

mOSCSetPBDIV() - Sets the peripheral bus divisor value. This is used to keep the Peripheral Bus clock under the maximum rate frequency or to set a lower peripheral bus frequency to save power.

OSCConfig() - Selects the desired clock source, the PLL multiplier, PLL postscaler, and the FRC divisor. Parameters not relevant to the desired clock source are written but have no effect and can be set to 0.

To avoid exceeding the maximum allowed frequency for the Peripheral Bus the order of operations for setting the PBBDIV divisor and the CPU must be chosen carefully. In general when switching to a higher CPU clock frequency the Peripheral Bus divisor should be set to the new lower value before changing the CPU frequency.

8.1 Individual Functions

OSCConfig()

**Description:** This sets the desired oscillator source, PLL postscaler, PLL multiplier and FRC divisor values.

**Include:** plib.h

**Prototype:**

```c
void OSCConfig(unsigned long int config1,
               unsigned long int config2,
               unsigned long int config3,
               unsigned long int config4);
```

**Arguments:**

- `config1` - This contains the bit field for the desired clock selection:

  - Osc Source Mode Select
  - OSC_FRC_DIV
  - OSC_FRC_DIV16
  - OSC_LPFC
  - OSC_SOSC
  - OSC_POSC_PLL
  - OSC_POSC
  - OSC_FRC_PLL
  - OSC_FRC

  (These bit fields are mutually exclusive)

- `config2` - This contains the bit field for the desired PLL multiplier selection.
8.2 Individual Macros

mOSCClockFailStatus()

Description: This macro returns the Clock Fail status.
Include: plib.h
Prototype: unsigned int mOSCClockFailStatus(void);
Arguments: None
Return Value: 1 = A clock failure has been detected. 0 = A clock failure has not been detected
Remarks: None

OSCConfig() (Continued)

Osc PLL Multiplier value
OSC_PLL_MULT_15
OSC_PLL_MULT_16
OSC_PLL_MULT_17
OSC_PLL_MULT_18
OSC_PLL_MULT_19
OSC_PLL_MULT_20
OSC_PLL_MULT_21
OSC_PLL_MULT_24
(These bit fields are mutually exclusive)

config3 This contains the bit field for the desired PLL postscaler selection.

Osc PLL Postscaler value
OSC_PLL_POST_1
OSC_PLL_POST_2
OSC_PLL_POST_4
OSC_PLL_POST_8
OSC_PLL_POST_16
OSC_PLL_POST_32
OSC_PLL_POST_64
OSC_PLL_POST_256
(These bit fields are mutually exclusive)

Arguments: config4 This contains the bit field for the desired FRC divisor selection.

Osc FRC divisor value
OSC_FRC_DIV_1
OSC_FRC_DIV_2
OSC_FRC_DIV_4
OSC_FRC_DIV_8
OSC_FRC_DIV_16
OSC_FRC_DIV_32
OSC_FRC_DIV_64
OSC_FRC_DIV_256
(These bit fields are mutually exclusive)

Return Value: None
Remarks: This function switches to FRC and then to the desired Source. Any parameters that are not relevant to the desired clock source can be set to 0. Interrupts must be disabled.

Code Example:
OscConfig(OSC_POSC_PLL, OSC_PLL_MULT_15,
OSC_PLL_POST_1, 0);

mOSCClockFailStatus()
**mOSCClockFailStatus()** (Continued)

**Code Example:**
```c
unsigned int result;
result = mOSCClockFailStatus();
```

---

### mOSCDisableSOSC()

**Description:** This macro disables the Secondary Oscillator (SOSC).

**Include:** `plib.h`

**Prototype:** `void mOSCDisableSOSC(void);`

**Arguments:** None

**Return Value:** None

**Remarks:** Interrupts must be disabled

**Code Example:**
```c
mOSCDisableSOSC();
```

---

### mOSCEnableSOSC()

**Description:** This macro enables the Secondary Oscillator (SOSC).

**Include:** `plib.h`

**Prototype:** `void mOSCEnableSOSC(void);`

**Arguments:** None

**Return Value:** None

**Remarks:** Interrupts must be disabled

**Code Example:**
```c
mOSCEnableSOSC();
```
### mOSCGetPBDIV()

**Description:** This macro returns the Peripheral Bus divisor.

**Include:** plib.h

**Prototype:**

```c
mOSCGetPBDIV();
```

**Arguments:** None

**Return Value:**

<table>
<thead>
<tr>
<th>Osc Source Mode Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - divisor is 1</td>
</tr>
<tr>
<td>1 - divisor is 2</td>
</tr>
<tr>
<td>2 - divisor is 4</td>
</tr>
<tr>
<td>3 - divisor is 8</td>
</tr>
</tbody>
</table>

**Remarks:** None

**Code Example:**

```c
unsigned long int divisor;
divisor = mOscGetPBDIV();
```

### mOSCSetPBDIV()

**Description:** This macro sets the Peripheral Bus divisor.

**Include:** plib.h

**Prototype:**

```c
mOSCSetPBDIV(unsigned int config);
```

**Arguments:**

<table>
<thead>
<tr>
<th>config</th>
<th>This contains the bit field for the desired clock selection:</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSC_PB_DIV_1</td>
<td>Osc Source Mode Select</td>
</tr>
<tr>
<td>OSC_PB_DIV_2</td>
<td></td>
</tr>
<tr>
<td>OSC_PB_DIV_4</td>
<td></td>
</tr>
<tr>
<td>OSC_PB_DIV_8</td>
<td></td>
</tr>
</tbody>
</table>

(These bit fields are mutually exclusive)

**Return Value:** None

**Remarks:** Interrupts must be disabled

**Code Example:**

```c
mOscSetPBDIV(OSC_PB_DIV_8);
```
8.3 Example of Use

// Master header file for all peripheral library includes
#include <plib.h>

main()
{
    // this example sets the cpu clock to FRC and then to POSC PLL
    OscConfig(OSC_FRC, 0, 0, 0); // set CPU clock to FRC
    mOSCSetPBIV(OSC_PB_DIV_4); // set PBIV before switching to the
    // faster clock source to prevent violating
    // PBCLK timing requirements
    OscConfig(OSC_POSC_PLL, OSC_PLL_MULT_15, OSC_PLL_POST_1, 0);
}
9.0  POWER SAVE FUNCTIONS

The PIC32MX has two power save modes: Sleep and Idle. The power save library macros are available to allow high-level control of these modes. The following macros are available:

mPowerSaveIdle() - Configures the device for Idle mode and enters Idle
mPowerSaveSleep() - Configures the device for Sleep mode and enters Sleep

9.1  Individual Functions

There are no functions to support this module, refer to the macro section

9.2  Individual Macros

mPowerSaveIdle()

| Description | This function places the CPU in Idle mode. |
| Include:    | plib.h                                      |
| Prototype:  | mPowerSaveIdle();                          |
| Arguments:  | None                                       |
| Return Value: | None                                    |
| Source File: | plib.h                                    |
| Remarks:    |                                            |
| Code Example: | mPowerSaveIdle();                        |

mPowerSaveSleep()

| Description | This function places the CPU in Sleep mode. |
| Include:    | plib.h                                      |
| Prototype:  | mPowerSaveSleep();                         |
| Arguments:  | None                                       |
| Return Value: | None                                    |
| Source File: | plib.h                                    |
| Remarks:    |                                            |
| Code Example: | mPowerSaveSleep();                        |

// Master header file for all peripheral library includes
#include <plib.h>

main()
{
    // this example puts the CPU in Sleep
    mPowerSaveSleep();
}
mPowerSaveSleep(); // configure for and enter sleep
10.0 I/O PORT LIBRARY

The PIC32MX I/O PORT library consists of simple, code efficient macros and functions supporting common control features for this peripheral. Several functions and macros have a similar name, but differ by the level of control they provide, Advanced or Basic.

Depending on the application, the advanced functions may provide greater flexibility compared to the similarly named basic macros, however, at the cost of slightly less efficient code due to overhead involved when calling any function. The basic macros can generate more efficient “compile-time” code. For specific details regarding their operations, refer to the function and macro descriptions in the following I/O Port sections.

*Note: some library features are “legacy” 16-Bit peripheral macros or functions and are maintained to provide compatibility for 16-Bit to 32-Bit PIC32MX code migration.

FUNCTION AND MACROS

The following function and macro categories are available:
• DIGITAL PIN CONFIGURATION
• ANALOG PIN CONFIGURATION
• INPUT/OUTPUT PIN DIRECTION
• OPEN DRAIN CONFIGURATION
• CHANGE NOTICE AND WEAK PULLUP CONFIGURATION
• EXTERNAL INTERRUPT PIN CONFIGURATION
• READ OPERATIONS
• WRITE OPERATIONS
• MISC OPERATIONS

FUNCTION AND MACRO PARAMETERS

Most function and macro parameters are simple bit mask symbols defined in the PORTS.h header file. One or more bit mask symbols may be bitwise OR’d together to select multiple bits.

For example: `mPORTASetBits(BIT_8 | BIT_10)`

Note: An absent bit mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

Some functions use an enumeration type to specify the applicable PORT.

For example: `PORTSetBits(IOPORT_A, BIT_8 | BIT_10)`
10.1 DIGITAL PIN CONFIGURATION

**Macros**

- `mPORTASetPinsDigitalIn()` ...
- `mPORTGSetPinsDigitalIn()`
- `mPORTASetPinsDigitalOut()` ...
- `mPORTGSetPinsDigitalOut()`

**Functions**

- `PORTSetPinsDigitalIn()`
- `PORTSetPinsDigitalOut()`

**Description**

Before reading and writing to any I/O port, the data direction of a desired pin must be properly configured as digital input or digital output. Some port I/O pins share digital and analog features and require the analog feature to be disabled when configuring the I/O port pin for digital mode.

**Usage**

These functions are typically used early in the program execution to establish the proper mode and state of the general purpose I/O pins for use as digital inputs and outputs.

**Advanced**

These functions configure port pins as digital input or digital output and automatically disable analog features that may be multiplexed with the specified pin(s).

**Feature:** Complete digital I/O pin configuration. These functions meet all the necessary configuration requirements to properly configure port I/O pins that are digital only and those port I/O pins that share analog and digital functionality.

**When to use:** These functions provide a simple and preferred method to configure digital I/O pins when the user is not familiar with the details of an I/O port. The user only needs to specify a `PORT` and `PIN(s)`.

For example: `PORTSetPinsDigitalIn(IOPORT_B, BIT_0)`

```
PORTSetPinsDigitalIn()
PORTSetPinsDigitalOut()
```

**Basic**

These macros configure port pins as digital input or digital output.

**Feature:** Simple digital I/O pin configuration only.

**When to use:** These macros provide basic digital I/O pin configuration when the control of other I/O port aspects or code efficiency is a desire. It is recommended that the user be familiar with the detailed I/O PORT operation. The user is responsible for disabling any analog input that may be multiplexed with the specified pin. The user only needs to specify the `PIN(s)`.

For example: `mPORTASetPinsDigitalIn(BIT_0)`

```
mPORTASetPinsDigitalIn()
...
mPORTGSetPinsDigitalIn()
mPORTASetPinsDigitalOut()
...
mPORTGSetPinsDigitalOut()
```
PORTSetPinsDigitalIn

Description: This function configures PORTx pins as digital inputs.

Include: plib.h

Prototype: void PORTSetPinsDigitalIn(IO_PORT_ID port, unsigned int inputs);

Arguments: port This argument is an IO_PORT_ID which specifies the desired port. Select only one mask from the mask set defined below.

IO PORT ID
IOPORT_A
IOPORT_B
IOPORT_C
IOPORT_D
IOPORT_E
IOPORT_F
IOPORT_G

inputs This argument contains one or more bit masks bitwise OR'd together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

IO Pin Bit Masks
BIT_0
BIT_1
BIT_2
...
BIT_15

Return Value: None

Remarks: For those IO pins that share digital and analog functionality, the corresponding ADPCFG bits are set appropriately. See code example.

Source File: port_set_pins_digital_in_lib.c

Code Example:
#define PORT IOPORT_C
#define PINS BIT_1 | BIT_0
PORTSetPinsDigitalIn(PORT, PINS);
PORTSetPinsDigitalOut

Description: This function configures PORTx pins as digital outputs.

Include: plib.h

Prototype: void PORTSetPinsDigitalOut(IO_PORT_ID port, unsigned int inputs);

Arguments: 

- **port**: This argument is an IO_PORT_ID which specifies the desired port. Select only one mask from the mask set defined below.

- **inputs**: This argument contains one or more bit masks bitwise OR'd together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

IO PORT ID
- IOPORT_A
- IOPORT_B
- IOPORT_C
- IOPORT_D
- IOPORT_E
- IOPORT_F
- IOPORT_G

IO Pin Bit Masks
- BIT_0
- BIT_1
- BIT_2
- ...
- BIT_15

Return Value: None

Remarks: For those IO pins that share digital and analog functionality, the corresponding ADPCFG bits are set appropriately. See code example.

Source File: port_set_pins_digital_out_lib.c

Code Example:
```c
#define PORT  IOPORT_B
#define PINS  BIT_7

PORTSetPinsDigitalOut(PORT, PINS);
```
mPORTASetPinsDigitalIn
mPORTBSetPinsDigitalIn
mPORTCSetPinsDigitalIn
mPORTDSetPinsDigitalIn
mPORTESetPinsDigitalIn
mPORTFSetPinsDigitalIn
mPORTGSetPinsDigitalIn
mPORTHSetPinsDigitalIn

Description:
This macro configures the TRISx register bits as inputs.

Include:
plib.h

Prototype:
void mPORTASetPinsDigitalIn(unsigned int _bits);
void mPORTBSetPinsDigitalIn(unsigned int _bits);
void mPORTCSetPinsDigitalIn(unsigned int _bits);
void mPORTDSetPinsDigitalIn(unsigned int _bits);
void mPORTESetPinsDigitalIn(unsigned int _bits);
void mPORTFSetPinsDigitalIn(unsigned int _bits);
void mPORTGSetPinsDigitalIn(unsigned int _bits);

Arguments:
_bits
This argument contains one or more bit masks bitwise OR’d together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

   IO Pin Bit Masks
   BIT_0
   BIT_1
   BIT_2
   ...
   BIT_15

Return Value:
None

Remarks:
Argument is copied to the TRISSETx register. If a bit is = ‘1’, the corresponding IO pin becomes an input; if a bit = ‘0’, the corresponding IO pin is not affected.
For those IO pins that share digital and analog functionality, the corresponding ADPCFG bits are set appropriately by the macro. See code example.

Same as mPORTxConfigInput

Source File:
None

Code Example:
/*PORTC<1:0> = inputs */
PORTCSetPinsDigitalIn(BIT_1 | BIT_0);

/* PORTA<8> inputs, all others not affected */
mPORTASetPinsDigitalIn(0x0100);
mPORTASetPinsDigitalOut
mPORTBSetPinsDigitalOut
mPORTCSetPinsDigitalOut
mPORTDSetPinsDigitalOut
mPORTESetPinsDigitalOut
mPORTFSetPinsDigitalOut
mPORTGSetPinsDigitalOut

Description: This macro configures the TRISx register bits as outputs.

Include:
plib.h

Prototype:
void mPORTASetPinsDigitalOut(unsigned int _bits);
void mPORTBSetPinsDigitalOut(unsigned int _bits);
void mPORTCSetPinsDigitalOut(unsigned int _bits);
void mPORTDSetPinsDigitalOut(unsigned int _bits);
void mPORTESetPinsDigitalOut(unsigned int _bits);
void mPORTFSetPinsDigitalOut(unsigned int _bits);
void mPORTGSetPinsDigitalOut(unsigned int _bits);

Arguments: _bits
This argument contains one or more bit masks bitwise OR’d together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

IO Pin Bit Masks
BIT_0
BIT_1
BIT_2
...
BIT_15

Return Value: None

Remarks: Argument is copied to the TRISCLRx register. If a bit is = ‘1’, the corresponding IO pin becomes an output; if a bit = ‘0’, the corresponding IO pin is not affected.
For those IO pins that share digital and analog functionality, the corresponding ADPCFG bits are set appropriately by the macro. See code example.

Same as mPORTxConfigOutput

Source File: None

Code Example:
/* make PORTE<7:6> = outputs */
mPORTESetPinsDigitalOut(BIT_7 | BIT_6);

/* PORTD<3> = output, all others not affected */
mPORTDSetPinsDigitalOut(0x0008);
10.2 ANALOG PIN CONFIGURATION

• Description

Before applying any analog input voltage or enabling an analog output peripheral on those
I/O port pins that are analog capable, typically PORTB only, the data direction of a desired
pin must be properly configured as analog input or analog output. Some port I/O pins
share digital and analog features and require the digital feature to be disabled when con-
figuring the I/O port pin for analog mode. Note, on Power-on Reset, analog is the default
mode for those I/O port pins that share digital and analog features.

• Useage

These functions are typically used early in the program execution to establish the proper
mode and state of the general purpose I/O pins for use as analog inputs and outputs.

• Advanced

These functions configure port pins as analog input or analog output and automatically
disable digital features that may be multiplexed with the specified pin(s).

Feature: Complete analog I/O pin configuration. These functions meet all the necessary
configuration requirements to properly configure port I/O pins that share analog and dig-
ital functionality.

When to use: These functions provide a simple and preferred method to configure ana-
log I/O pins when the user is not familiar with the details of an I/O port. The user only
needs to specify a PORT and PIN(s).

For example: PORTSetPinsAnalogIn(IOPORT_B, BIT_0)

PORTSetPinsAnalogIn
PORTSetPinsAnalogOut

• Basic

These macros configure port pins as analog input or analog output.

Feature: Simple analog I/O pin configuration only.

When to use: These macros provide basic analog I/O pin configuration when the control
of other I/O port aspects or code efficiency is a desire. It is recommended that the user be
familiar with the detailed I/O PORT operation. The user is responsible for disabling any
analog input that may be multiplexed with the specified pin. The user only needs to specify
the PIN(s).

For example: mPORTBSetPinsDigitalIn(BIT_0)

mPORTBSetPinsAnalogIn
mPORTBSetPinsAnalogOut
PORTSetPinsAnalogIn

**Description:**
This function configures PORTx pins as analog inputs.

**Include:**
plib.h

**Prototype:**
void PORTSetPinsAnalogIn(IO_PORT_ID port, unsigned int inputs);

**Arguments:**
- **port**
  This argument is an IO_PORT_ID which specifies the desired port. Select only one mask from the mask set defined below.

  **IO PORT ID**
  - IOPORT_A
  - IOPORT_B
  - IOPORT_C
  - IOPORT_D
  - IOPORT_E
  - IOPORT_F
  - IOPORT_G

- **inputs**
  This argument contains one or more bit masks bitwise OR’d together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

  **IO Pin Bit Masks**
  - BIT_0
  - BIT_1
  - BIT_2
  - ...
  - BIT_15

**Return Value:**
None

**Remarks:**
For those IO pins that share digital and analog functionality, the corresponding ADPCFG bits are set appropriately. See code example.

**Source File:**
port_set_pins_analog_in_lib.c

**Code Example:**
```
#define PORT  IOPORT_B
#define PINS  BIT_1 | BIT_0
PORTSetPinsAnalogIn(PORT, PINS);
```
PORTSetPinsAnalogOut

Description: This function configures PORTx pins as digital outputs.

Include: plib.h

Prototype: void PORTSetPinsAnalogOut(IO_PORT_ID port, unsigned int inputs);

Arguments:

port This argument is an IO_PORT_ID which specifies the desired port. Select only one mask from the mask set defined below.

IO PORT ID
IOPORT_A
IOPORT_B
IOPORT_C
IOPORT_D
IOPORT_E
IOPORT_F
IOPORT_G

inputs This argument contains one or more masks bitwise OR’d together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

IO Pin Bit Masks
BIT_0
BIT_1
BIT_2
...
BIT_15

Return Value: None

Remarks: For those IO pins that share digital and analog functionality, the corresponding ADPCFG bits are set appropriately. See code example.

Source File: port_set_pins_analog_out_lib.c

Code Example:
#define PORT IOPORT_B
#define PINS BIT_10

PORTSetPinsAnalogOut(PORT, PINS);
mPORTBSetPinsAnalogIn

Description: This macro configures the TRISB register bits as inputs and corresponding ADPCFG register bits as analog.

Include: pplib.h

Prototype: void mPORTBSetPinsAnalogIn(unsigned int _bits);

Arguments: _bits This argument contains one or more bit masks bitwise OR'd together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

IO Pin Bit Masks
BIT_0
BIT_1
BIT_2
...
BIT_15

Return Value: None

Remarks: Argument is copied to the TRISSETB register. If a bit is = ‘1’, the corresponding IO pin becomes an input; if a bit = ‘0’, the corresponding IO pin is not affected.

For those IO pins that share digital and analog functionality, the corresponding ADPCFG bits are set by the macro. See code example.

Source File: None

Code Example: /*PORTB<1:0> = analog inputs */
mPORTBSetPinsAnalogIn(BIT_1 | BIT_0);
mPORTBSetPinsAnalogOut

Description: This macro configures the TRISB register bits as outputs and corresponding ADPCFG register bits as analog.

Include: plib.h

Prototype: void mPORTBSetPinsAnalogOut(unsigned int _bits);

Arguments: _bits

This argument contains one or more bit masks bitwise OR’d together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

IO Pin Bit Masks
BIT_0
BIT_1
BIT_2
...
BIT_15

Return Value: None

Remarks: Argument is copied to the TRISCLRB register. If a bit is = ‘1’, the corresponding IO pin becomes an output; if a bit = ‘0’, the corresponding IO pin is not affected.
For those IO pins that share digital and analog functionality, the corresponding ADPCFG bits are set appropriately by the macro. See code example.

Source File: None

Code Example: /* make PORTB<10> = (CVref) analog output */
mPORTBSetPinsAnalogOut(BIT_10);
10.3 INPUT/OUTPUT PIN DIRECTION

Macros

mPORTADirection() ...
mPORTDDirection() ...
mPORTAGetDirection() ...
mPORTGGetDirection() ...
mPORTAReadDirectionBits() ...
mPORTGReadDirectionBits() ...
mPORTACloseBits() ...
mPORTGCloseBits() ...
mPORTACloseAll() ...
mPORTGCloseAll() 

- Description
At Power-On Reset, all I/O pins default to inputs. Before reading and writing to any I/O port, the data direction of an I/O pin must be properly configured as input or output.

- Usage
These functions are typically used early in the program execution to establish the desired direction of the general purpose IO pins. Macros mPORTxDirection(), mPORTxGetDirection() and mCloseAll() operate directly on the TRIS register and therefore modify the entire register with the contents of the argument. Macros mPORTxCloseBits() and mPORTAReadDirectionBits() will only affect those bits specified in the argument.

Note: To specify input and output direction on specific pins without affecting neighboring pin configuration on the target port, use macros mPORTxSetPinsDigitalIn() or mPORTxSetPinsDigitalOut().

- Basic
These macros configure port pin directions.

Feature: Simple I/O pin direction configuration only.

When to use: Use these macros to configure a port’s direction (TRIS) register. It is recommended that the user is familiar with the detailed IO PORT operation. The user is responsible for disabling any analog input that may be multiplexed with the specified pin. The user only needs to specify the PIN(s).

For example: mPORTFDirection(BIT_7)
mPORTADirection() ...
...mPORTDDirection() 
mPORTAGetDirection() ...
...mPORTGGetDirection() 
mPORTAReadDirectionBits() ...
...mPORTGReadDirectionBits() 
mPORTACloseBits() ...
...mPORTGCloseBits() 
mPORTACloseAll() ...
...
mPORTADirection
mPORTBDirection
mPORTCDirection
mPORTDDirection
mPORTEDirection
mPORTFDirection
mPORTGDirection

Description: This macro configures the complete TRISx register. Both inputs and outputs are specified in the argument.

Include: plib.h

Prototype:
void mPORTADirection(unsigned int _bits);
void mPORTBDirection(unsigned int _bits);
void mPORTCDirection(unsigned int _bits);
void mPORTDDirection(unsigned int _bits);
void mPORTEDirection(unsigned int _bits);
void mPORTFDirection(unsigned int _bits);
void mPORTGDirection(unsigned int _bits);

Arguments: _bits This argument contains one or more bit masks bitwise OR’d together. Select one or more mask bits from the mask set defined below to configure a corresponding pin as an input. An absent mask symbol configures corresponding bit(s) as an output and will be set = 0.

IO Pin Bit Masks
BIT_0
BIT_1
BIT_2
...
BIT_15

Return Value: None

Remarks: Argument is copied to the TRISx register, therefore all bits are modified. If a bit is ‘1’, the corresponding IO pin becomes an input; if a bit = ‘0’, the corresponding IO pin becomes an output. See code example.

Source File: None

Code Example:
/* PORTC<1:0> = inputs, all others outputs */
mPORTCDirection(BIT_1 | BIT_0);

/* PORTB<1>,<5:4> = inputs, all others outputs */
mPORTBDirection(0x0032);
### mPORTACloseBits
- **Description:** This macro sets the specified IO Port pin as input and clears its corresponding LATx register bit.
- **Include:** `plib.h`
- **Prototype:**
  ```c
  void mPORTACloseBits(unsigned int _bits);
  ```
- **Arguments:** `_bits` This argument contains one or more bit masks bitwise OR'd together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.
- **IO Pin Bit Masks**
  - BIT_0
  - BIT_1
  - BIT_2
  - ...
  - BIT_15
- **Return Value:** None
- **Remarks:** To close a specific IO pin, include its bit mask in the argument.
  If a mask bit is = '1', the corresponding IO pin is set as an input and the corresponding LATx bit is set = 0; if a mask bit = '0', the corresponding IO pin is not affected.
- **Source File:** None
- **Code Example:**
  ```c
  /* close PORTF<5,3,1> bits */
  mPORTFCloseBits(BIT_5 | BIT_3 | BIT_1);
  ```
mPORTACloseAll
mPORTBCloseAll
mPORTCCloseAll
mPORTDCloseAll
mPORTECloseAll
mPORTFCloseAll
mPORTGCloseAll

Description: This macro sets all IO Port pins as input and clears their corresponding LATx register bits.

Include: plib.h

Prototype:
void mPORTACloseAll(void);
void mPORTBCloseAll(void);
void mPORTCCloseAll(void);
void mPORTDCloseAll(void);
void mPORTECloseAll(void);
void mPORTFCloseAll(void);
void mPORTGCloseAll(void);

Arguments: None

Return Value: None

Remarks: See code example

Source File: None

Code Example:
/* close PORTA */
mPORTACloseAll();
mPORTAGetDirection
mPORTBGetDirection
mPORTCGetDirection
mPORTDGetDirection
mPORTEGetDirection
mPORTFGetDirection
mPORTGGetDirection

Description: This macro provides the contents of TRISx register.

Include: plib.h

Prototype:
void mPORTAGetDirection(void);
void mPORTBGetDirection(void);
void mPORTCGetDirection(void);
void mPORTDGetDirection(void);
void mPORTEGetDirection(void);
void mPORTFGetDirection(void);
void mPORTGGetDirection(void);

Arguments: None

Remarks: Same as reading the TRISx register. See code example.

Source File: None

Code Example:
/* get the configuration of TRISC */
config = mPORTCGetDirection();
mPORTAReadDirectionBits
mPORTBReadDirectionBits
mPORTCReadDirectionBits
mPORTDReadDirectionBits
mPORTEReadDirectionBits
mPORTFReadDirectionBits
mPORTGReadDirectionBits

Description: This macro provides the masked contents of TRISx register.

Include: plib.h

Prototype:
unsigned int mPORTAReadDirectionBits
(unsigned int _bits);
unsigned int mPORTBReadDirectionBits
(unsigned int _bits);
unsigned int mPORTCReadDirectionBits
(unsigned int _bits);
unsigned int mPORTDReadDirectionBits
(unsigned int _bits);
unsigned int mPORTEReadDirectionBits
(unsigned int _bits);
unsigned int mPORTFReadDirectionBits
(unsigned int _bits);
unsigned int mPORTGReadDirectionBits
(unsigned int _bits);

Arguments: _bits This argument contains one or more bit masks bitwise OR’d together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

IO Pin Bit Masks
BIT_0
BIT_1
BIT_2
...
BIT_15

Return Value: None

Remarks: The bit mask is bitwise AND’d with the contents of the TRISx register. See code example

Source File: None

Code Example: /* get the configuration of bit 15 of TRISC */
config = mPORTCReadDirectionBits(BIT_15);
10.4 OPEN DRAIN CONFIGURATION

**Macros**

- `mPORTAOpenDrainOpen()` ...
- `mPORTGOpenDrainOpen()`
- `mPORTAOpenDrainClose()` ...
- `mPORTGOpenDrainClose()`

**Description**

Before reading and writing to any I/O port, the data direction of a desired pin must be properly configured as digital input or digital output. Some port I/O pins share digital and analog features and require the analog feature to be disabled when configuring the I/O port pin for digital mode.

**Usage**

These functions are typically used early in the program execution to establish the proper mode and state of the general purpose IO pins.

**Basic**

These macros configure port pins as digital input or digital output.

*Feature:* Simple digital IO pin configuration only.

*When to use:* These macros provide basic digital IO pin configuration for users who need to control other aspects of the port IO pin configuration and it is recommended that the user is familiar with the detailed IO PORT operation. The user is responsible for disabling any analog input that may be multiplexed with the specified pin. The user only needs to specify the PIN(s).

*For example:* `mPORTBSetPinsDigitalIn(BIT_0)`

```c
mPORTxOpenDrainOpen
mPORTxOpenDrainClose
```
mPORTAOpenDrainOpen
mPORTBOpenDrainOpen
mPORTCOpenDrainOpen
mPORTDOpenDrainOpen
mPORTEOpenDrainOpen
mPORTFOpenDrainOpen
mPORTGOpenDrainOpen

Description: This macro enables the IO Port pin open drain feature.

Include: plib.h

Prototype:
void mPORTAOpenDrainOpen(unsigned int _bits);
void mPORTBOpenDrainOpen(unsigned int _bits);
void mPORTCOpenDrainOpen(unsigned int _bits);
void mPORTDOpenDrainOpen(unsigned int _bits);
void mPORTEOpenDrainOpen(unsigned int _bits);
void mPORTFOpenDrainOpen(unsigned int _bits);
void mPORTGOpenDrainOpen(unsigned int _bits);

Arguments: _bits This argument contains one or more bit masks bitwise OR'd together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

IO Pin Bit Masks
BIT_0
BIT_1
BIT_2
...
BIT_15

Return Value: None

Remarks: To enable a specific IO pin as open-drain output, include its bit mask in the argument. If a mask bit is ‘1’, the corresponding TRISx bit is set = 0 (output) and corresponding IO pin open drain feature is enabled; if a mask bit = ‘0’, the corresponding IO pin is not affected. See code example

Source File: None

Code Example:
/* enable open drain outputs PORTE<7:6> */
mPORTEOpenDrainOpen(BIT_7 | BIT_6);
mPORTAOpenDrainClose
mPORTBOpenDrainClose
mPORTCOpenDrainClose
mPORTDOpenDrainClose
mPORTEOpenDrainClose
mPORTFOpenDrainClose
mPORTGOpenDrainClose

Description: This macro disables an IO Port pin open drain.

Include: plib.h

Prototype:
void mPORTAOpenDrainClose(unsigned int _bits);
void mPORTBOpenDrainClose(unsigned int _bits);
void mPORTCOpenDrainClose(unsigned int _bits);
void mPORTDOpenDrainClose(unsigned int _bits);
void PORTEOpenDrainClose(unsigned int _bits);
void PORTFOpenDrainClose(unsigned int _bits);
void PORTGOpenDrainClose(unsigned int _bits);

Arguments: _bits This argument contains one or more bit masks bitwise OR’d together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

IO Pin Bit Masks
BIT_0
BIT_1
BIT_2
...
BIT_15

Return Value: None

Remarks: To disable a specific IO pin open-drain output, include its bit mask in the argument. If a mask bit is = ‘1’, the corresponding TRISx bit is set = 1 (input) and corresponding IO pin open drain feature is disabled; if a mask bit = ‘0’, the corresponding IO pin is not affected. See code example.

Source File: None

Code Example:
/* disable open drain outputs PORTE<7:6> */
mPORTEOpenDrainClose(BIT_7 | BIT_6);
10.5 CHANGE NOTICE AND WEAK PULLUP CONFIGURATION

**Macros**

- `mCNOpen()`
- `mCNClose()`
- `mCNEnable()`
- `*ConfigIntCN()`
- `*EnableCN0() ...`  
- `*EnableCN21()`
- `*DisableCN0() ...`  
- `*DisableCN21()`
- `*ConfigCNPullups()`

**Description**

Before reading and writing to any I/O port, the data direction of a desired pin must be properly configured as digital input or digital output. Some port I/O pins share digital and analog features and require the analog feature to be disabled when configuring the I/O port pin for digital mode.

**Usage**

These functions are typically used early in the program execution to establish the proper mode and state of the general purpose IO pins.

**Basic**

These macros configure port pins as digital input or digital output.

*Feature*: Simple digital IO pin configuration only.

*When to use*: These macros provide basic digital IO pin configuration for users who need to control other aspects of the port IO pin configuration and it is recommended that the user is familiar with the detailed IO PORT operation. The user is responsible for disabling any analog input that may be multiplexed with the specified pin. The user only needs to specify the `PIN(s)`.

*For example*: `mPORTBSetPinsDigitalIn(BIT_0)`

```
*ConfigIntCN  
*EnableCNx  
*DisableCNx  
ConfigCNPullups
```

```
mCNOpen
mCNClose
mCNEnable
* = Legacy
```
**ConfigIntCN**

*Description:* This legacy macro sets the priority level for the Change Notice pins.

*Include:* plib.h

*Prototype:* void ConfigIntCN(unsigned int _bits);

*Arguments:*

<table>
<thead>
<tr>
<th>_bits</th>
<th>This argument contains one or more bit masks bitwise OR'd together. Select only one mask from each of the two mask sets defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.</th>
</tr>
</thead>
</table>

**CN Interrupt Enable/Disable**

<table>
<thead>
<tr>
<th>CHANGE_INT_ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHANGE_INT_OFF</td>
</tr>
</tbody>
</table>

**CN Interrupt Priority Bit Masks**

| CHANGE_INT_PRI_0 |
| CHANGE_INT_PRI_1 |
| CHANGE_INT_PRI_2 |
| ... |
| CHANGE_INT_PRI_7 |

*Return Value:* None

*Remarks:* Change notice interrupt flag is cleared, priority level is set and interrupt is enabled.

*Note:* Not all IO pins provide a change notice interrupt feature. Refer to the specific PIC32MX datasheet regarding the IO pins that support the change notice feature.

*See code example.*

*Source File:* None

*Code Example:* /* enable pullups on change notice pins 5 and 4 */

```
ConfigIntCN(CHANGE_INT_ON | CHANGE_INT_PRI_2);
```
*EnableCN0
*EnableCN1
*EnableCN2
...  
*EnableCN21

| Description | These legacy macros enable individual interrupt on change pins. |
| Include     | plib.h |
| Prototype   | void EnableCN0(void);
|             | void EnableCN1(void);
|             | void EnableCN2(void);
|             | ...
|             | void EnableCN21(void);
|             | void EnableCN_ALL(void); |
| Arguments   | None |
| Return Value| None |
| Remarks     | Sets the corresponding bit in CNENSET register. Not all IO pins provide a interrupt on change notice feature. Refer to the device’s datasheet regarding which IO pins provide interrupt on change notice. See code example. |
| Source File | None |
| Code Example| /* enable change notice pins 5 and 4 */
|             | EnableCN4;
|             | EnableCN5; |
*DisableCN0
*DisableCN1
*DisableCN2
...
*DisableCN21

Description: These legacy macros disable individual interrupt on change pins.
Include: plib.h
Prototype:
void DisableCN0 (void);
void DisableCN1 (void);
void DisableCN2 (void);
...
void DisableCN21 (void);
void DisableCN_ALL (void);

Arguments: None
Return Value: None
Remarks: Sets the corresponding bit in CNENCLR register.
Not all IO pins provide a interrupt on change notice feature. Refer to the device's datasheet regarding which IO pins provide interrupt on change notice. See code example.
Source File: None
Code Example:
/* disable on change notice pins 5 and 4 */
DisableCN4;
DisableCN5;
**ConfigCNPullups**

<table>
<thead>
<tr>
<th>Description:</th>
<th>This legacy macro enables individual pin pullups.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>plib.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td>void ConfigCNPullups(unsigned int _bits);</td>
</tr>
<tr>
<td>Arguments:</td>
<td>_bits This argument contains one or more bit masks, bitwise OR’d together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.</td>
</tr>
</tbody>
</table>
| CN Pullup Bit Masks | CN0_PULLUP_ENABLE  
CN1_PULLUP_ENABLE  
CN2_PULLUP_ENABLE  
...  
CN21_PULLUP_ENABLE  
CN_PULLUP_DISABLE_ALL |
| Return Value: | None |
| Remarks:     | Not all IO pins provide a interrupt on change pullup feature. Refer to the device’s datasheet regarding which IO pins provide interrupt on change pullup. See code example. |
| Source File: | None |
| Code Example: | /* enable pullups on change notice pins 10,11 */  
ConfigCNPullups(CN10_PULLUP_ENABLE |  
CN11_PULLUP_ENABLE); |
### mCNOpen

**Description:** This macro configures the change notice pins and the associated pullups.

**Include:** `plib.h`

**Prototype:**
```c
void mCNOpen(unsigned int _config, unsigned int _pins, unsigned int _pullups);
```

**Arguments:**
- `_config` This argument contains one or more bit masks bitwise OR'd together. Select only one mask from each of the three mask sets defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.
  - **CN module On/Off**
    - CN_ON
    - CN_OFF
  - **CN debug freeze mode On/Off**
    - CN_FRZ_ON
    - CN_FRZ_OFF
  - **CN idle mode On/Off**
    - CN_IDLE_CON
    - CN_IDLE_STOP
- `_pins` This argument contains one or more bit masks bitwise OR'd together. Select one or more from the following bit masks. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.
  - **CN Enable Pins**
    - CN0_ENABLE
    - CN1_ENABLE
    - CN2_ENABLE
    - ... 
    - CN21_ENABLE
    - CN_DISABLE_ALL
- `_pullups` This argument contains one or more bit masks bitwise OR'd together. Select one or more from the following bit masks. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.
  - **CN Enable Pullups**
    - CN0_PULLUP_ENABLE
    - CN1_PULLUP_ENABLE
    - CN2_PULLUP_ENABLE
    - ... 
    - CN21_PULLUP_ENABLE
    - CN_PULLUP_DISABLE_ALL

**Return Value:** None

**Notes:** An absent mask symbol in an argument assumes the corresponding bit(s) are disabled, or default value, and are set = 0.
Remarks:
Not all IO pins provide a interrupt on change pullup feature. Refer to the device’s datasheet regarding which IO pins provide interrupt on change pullup.

Note: To prevent spurious change notice interrupts during configuration, it is recommended to disable vector interrupts prior to configuring the change notice module, read the corresponding ports to clear any mismatch condition, enable change notice interrupts then re-enable vector interrupts.

See code example.

Source File: None

Code Example:
#define CONFIG (CN_ON | CN_IDLE_CON)
#define PINS (CN15_ENABLE)
#define PULLUPS (CN_PULLUP_DISABLE_ALL)
#define INTERRUPT (CHANGE_INT_ON | CHANGE_INT_PRI_2)

/* STEP 1. disable multi-vector interrupts */
mINTDisableSystemMultiVectoredInt();

/* STEP 2. setup the change notice options */
mCNOpen(CONFIG, PINS, PULLUPS);

/* STEP 3. read port(s) to clear mismatch */
value = mPORTDRead();
...

/* STEP 4. clear change notice interrupt flag */
mCNIntEnable(INTERRUPT);

/* STEP 5. enable multi-vector interrupts */
INTEnableSystemMultiVectoredInt();
### mCNClose

**Description:** This macro enables the specified on interrupt change pin pullups.

**Include:** `plib.h`

**Prototype:**

```c
void mCNClose(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** See code example.

**Source File:** None

**Code Example:**

```c
/* disable all change notice pins */
mCNClose();
```

---

### mCNEnable

**Description:** This macro enables one or more change notice pins.

**Include:** `plib.h`

**Prototype:**

```c
void mCNEnable(unsigned int _bits);
```

**Arguments:**

- `_bits` This argument contains one or more bit masks, bitwise OR'd together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

<table>
<thead>
<tr>
<th>CN Enable Bit Masks</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN0_ENABLE</td>
</tr>
<tr>
<td>CN1_ENABLE</td>
</tr>
<tr>
<td>CN2_ENABLE</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>CN21_ENABLE</td>
</tr>
</tbody>
</table>

**Return Value:** None

**Remarks:** Not all IO pins provide a interrupt on change pullup feature. Refer to the device’s datasheet regarding which IO pins provide interrupt on change pullup. See code example.

**Source File:** None

**Code Example:**

```c
/* enable pullups on change notice pins 5 and 4 */
mCNEnable(CN2_ENABLE | CN7_ENABLE | CN10_ENABLE);
```
10.6 EXTERNAL INTERRUPT PIN CONFIGURATION

Macros
SetPriorityINT0( ) ...
SetPriorityINT4( )
SetSubPriorityINT0( ) ...
SetSubPriorityINT4( )
*ConfigINT0( ) ...
*ConfigINT4( )
*CloseINT0( ) ...
*CloseINT4( )
*EnableINT0( ) ...
*EnableINT4( )
*DisableINT0( ) ...
*DisableINT4( )

• Description
Before reading and writing to any I/O port, the data direction of a desired pin must be properly configured as digital input or digital output. Some port I/O pins share digital and analog features and require the analog feature to be disabled when configuring the I/O port pin for digital mode.

• Usage
These functions are typically used early in the program execution to establish the proper mode and state of the general purpose IO pins.

• Basic
These macros configure port pins as digital input or digital output.

Feature: Simple digital IO pin configuration only.

When to use: These macros provide basic digital IO pin configuration for users who need to control other aspects of the port I/O pin configuration and it is recommended that the user is familiar with the detailed IO PORT operation. The user is responsible for disabling any analog input that may be multiplexed with the specified pin. The user only needs to specify the PIN(s).

For example: mPORTBSetsFlipIn(BIT_0)

*ConfigINTx
*CloseINTx
*EnableINTx
*DisableINTx

SetPriorityINTx
SetSubPriorityINTx
*ConfigINT0  
*ConfigINT1  
*ConfigINT2  
*ConfigINT3  
*ConfigINT4

**Description:** These legacy macros configure the external interrupts

**Include:** 
plib.h

**Prototype:**
```c
void ConfigInt0(unsigned int _bits);
void ConfigInt1(unsigned int _bits);
void ConfigInt2(unsigned int _bits);
void ConfigInt3(unsigned int _bits);
void ConfigInt4(unsigned int _bits);
```

**Arguments:**

>_bits_  
This argument contains one or more bit masks bitwise OR’d together. Select only one mask from each of the three mask sets defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

- **External Interrupt Enable/Disable**
  - EXT_INT_ENABLE
  - EXT_INT_DISABLE

- **External Interrupt Edge Detect**
  - RISING_EDGE_INT
  - FALLING_EDGE_INT

- **CN Interrupt Priority Bit Masks**
  - EXT_INT_PRI_0
  - EXT_INT_PRI_1
  - EXT_INT_PRI_2
  - ...
  - EXT_INT_PRI_7

**Return Value:** None

**Remarks:** Clears corresponding interrupt flag, configures the interrupt priority, external pin edge detect (rise/fall) and enables/disables the interrupt. See code example.

**Source File:** None

**Code Example:**
```c
/* configure external INT0 pin interrupt */
ConfigInt0(EXT_INT_ENABLE | RISING_EDGE_INT | EXT_INT_PRI_2);
```
*EnableINT0  
*EnableINT1  
*EnableINT2  
*EnableINT3  
*EnableINT4

**Description:** These legacy macros enable the specified external interrupt.  
**Include:** plib.h  
**Prototype:**
```c
void EnableInt0(void);
void EnableInt1(void);
void EnableInt2(void);
void EnableInt3(void);
void EnableInt4(void);
```

**Arguments:** None  
**Return Value:** None  
**Remarks:** See code example.  
**Source File:** None  
**Code Example:**
```c
/* enable external INT4 pin interrupt */
EnableINT4;
```

*DisableINT0  
*DisableINT1  
*DisableINT2  
*DisableINT3  
*DisableINT4

**Description:** These legacy macros disable the specified external interrupt.  
**Include:** plib.h  
**Prototype:**
```c
void DisableInt0(void);
void DisableInt1(void);
void DisableInt2(void);
void DisableInt3(void);
void DisableInt4(void);
```

**Arguments:** None  
**Return Value:** None  
**Remarks:** See code example.  
**Source File:** None  
**Code Example:**
```c
/* disable external INT4 pin interrupt */
DisableINT4;
```
*CloseINT0
*CloseINT1
*CloseINT2
*CloseINT3
*CloseINT4

Description: These legacy macros disable the specified external interrupt and clears interrupt flag.

Include: plib.h

Prototype:
void CloseInt0(void);
void CloseInt1(void);
void CloseInt2(void);
void CloseInt3(void);
void CloseInt4(void);

Arguments: None
Return Value: None
Remarks: INTx interrupt is disabled and corresponding interrupt flag is cleared.

Source File: None
Code Example:
/* closes external INT4 pin interrupt */
CloseINT4;
**SetPriorityINT0**  
**SetPriorityINT1**  
**SetPriorityINT2**  
**SetPriorityINT3**  
**SetPriorityINT4**

**Description:** These legacy macros set the priority level for the specified external interrupt pin.

**Include:** plib.h

**Prototype:**

```c
void SetPriorityInt0(unsigned int _bits);
void SetPriorityInt1(unsigned int _bits);
void SetPriorityInt2(unsigned int _bits);
void SetPriorityInt3(unsigned int _bits);
void SetPriorityInt4(unsigned int _bits);
```

**Arguments:**

- `_bits` This argument contains one bit mask. Select only one mask from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

**External Interrupt Priority Bit Masks**

```c
EXT_INT_PRI_0
EXT_INT_PRI_1
EXT_INT_PRI_2
...
EXT_INT_PRI_7
```

**Return Value:** None

**Remarks:** See code example.

**Source File:** None

**Code Example:**

```c
/* configure priority level 5 */
SetPriorityInt3(EXT_INT_PRI_5);
```
SetSubPriorityINT0
SetSubPriorityINT1
SetSubPriorityINT2
SetSubPriorityINT3
SetSubPriorityINT4

Description: These macros set the sub-priority level for the specified external interrupt pin.

Include: plib.h

Prototype: void SetSubPriorityINT0(unsigned int _bits);
void SetSubPriorityINT1(unsigned int _bits);
void SetSubPriorityINT2(unsigned int _bits);
void SetSubPriorityINT3(unsigned int _bits);
void SetSubPriorityINT4(unsigned int _bits);

Arguments: _bits This argument contains one bit mask. Select only one mask from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

External Interrupt Sub Priority Bit Masks
EXT_INT_SUB_PRI_0
EXT_INT_SUB_PRI_1
EXT_INT_SUB_PRI_2
EXT_INT_SUB_PRI_3

Return Value: None

Remarks: See code example.

Source File: None

Code Example: /* configure sub priority level 2 */
SetSubPriorityInt0(EXT_INT_SUB_PRI_2);
10.7 READ OPERATIONS

**Description**

Before reading and writing to any I/O port, the data direction of a desired pin must be properly configured as digital input or digital output. Some port I/O pins share digital and analog features and require the analog feature to be disabled when configuring the I/O port pin for digital mode.

**Usage**

These functions are typically used early in the program execution to establish the proper mode and state of the general purpose IO pins.

**Advanced**

These functions configure port pins as digital input or digital output and automatically disable analog inputs that may be multiplexed with the specified pin(s).

*Feature:* Complete digital IO pin configuration. These functions meet all the necessary configuration requirements to properly configure port IO pins that are digital only and those port IO pins that share analog and digital functionality.

*When to use:* These functions provide a simple and preferred method to configure digital I/O pins when the user is not familiar with the details of an I/O port. The user only needs to specify a *PORT* and *PIN(s)*.

*For example:* `PORTRead(IOPORT_B)`  
`PORTRead`  
`PORTReadBits`

**Basic**

These macros configure port pins as digital input or digital output.

*Feature:* Simple digital IO pin configuration only.

*When to use:* These macros provide basic digital IO pin configuration for users who need to control other aspects of the port IO pin configuration and it is recommended that the user is familiar with the detailed IO PORT operation. The user is responsible for disabling any analog input that may be multiplexed with the specified pin. The user only needs to specify the *PIN(s)*.

*For example:* `mPORTReadLatchBits(BIT_0)`

`mPORTxARead`  
`mPORTxAReadBits`  
`mPORTxAReadLatch`  
`mPORTxAReadLatchBits`  

**Macros**

- `mPORTARead( )` ...
- `mPORTGRead( )`
- `mPORTAReadBits( )` ...
- `mPORTGReadBits( )`
- `mPORTAReadLatch( )` ...
- `mPORTGReadLatch( )`
- `mPORTAReadLatchBits( )` ...
- `mPORTGReadLatchBits( )`

**Functions**

- `PORTRead( )`
- `PORTReadBits( )`
PORTRead

Description: This function reads and returns the contents of a specified PORT.

Include: plib.h

Prototype: unsigned int PORTRead(IO_PORT_ID port);

Arguments: port This argument is an IO_PORT_ID which specifies the desired port. Select only one mask from the mask set defined below.

IO PORT ID
IOPORT_A
IOPORT_B
IOPORT_C
IOPORT_D
IOPORT_E
IOPORT_F
IOPORT_G

Return Value: unsigned int = value read from specified PORT register

Remarks:

Source File: port_read_lib.c

Code Example:
/* read PORT C */
value = PORTRead(IOPORT_C);

PORTReadBits

Description: This function reads and returns only the specified bits from a specified PORT.

Include: plib.h

Prototype: unsigned int PORTReadBits(IO_PORT_ID port, unsigned int bits);

Arguments: port This argument is an IO_PORT_ID which specifies the desired port. Select only one mask from the mask set defined below.

IO PORT ID
IOPORT_A
IOPORT_B
IOPORT_C
IOPORT_D
IOPORT_E
IOPORT_F
IOPORT_G

_bits This argument contains one or more bit masks bitwise OR’d together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.
### PORTReadBits (Continued)

<table>
<thead>
<tr>
<th>IO Pin Bit Masks</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT_0</td>
</tr>
<tr>
<td>BIT_1</td>
</tr>
<tr>
<td>BIT_2</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>BIT_15</td>
</tr>
</tbody>
</table>

**Return Value:** unsigned int = value read from specified PORT register bitwise AND'd with _bits parameter.

**Remarks:**

**Source File:** port_read_bits_lib.c

**Code Example:**
```c
/* read PORT C */
value = PORTReadBits(IOPORT_C, BIT_7 | BIT_6);
```
mPORTARead
mPORTBRead
mPORTCRead
mPORTDRead
mPORTERead
mPORTFRead
mPORTGRead

Description: This macro provides the contents of PORTx register.
Include: plib.h
Prototype:
unsigned int mPORTARead(void);
unsigned int mPORTBRead(void);
unsigned int mPORTCRead(void);
unsigned int mPORTDRead(void);
unsigned int mPORTERead(void);
unsigned int mPORTFRead(void);
unsigned int mPORTGRead(void);

Arguments: None
Return Value: unsigned int = value read from specified PORTx register
Remarks: Same as reading the PORTx register. See code example
Source File: None
Code Example:
/* read PORT C */
value = mPORTCRead();
mPORTAReadBits
mPORTBReadBits
mPORTCReadBits
mPORTDReadBits
mPORTEReadBits
mPORTFReadBits
mPORTGReadBits

Description: This macro provides the masked contents of PORTx register.
Include: 
plib.h
Prototype: 
unsigned int mPORTAReadBits(unsigned int _bits);
unsigned int mPORTBReadBits(unsigned int _bits);
unsigned int mPORTCReadBits(unsigned int _bits);
unsigned int mPORTDReadBits(unsigned int _bits);
unsigned int mPORTEReadBits(unsigned int _bits);
unsigned int mPORTFReadBits(unsigned int _bits);
unsigned int mPORTGReadBits(unsigned int _bits);

Arguments: _bits This argument contains one or more bit masks bitwise OR’d together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

IO Pin Bit Masks
BIT_0
BIT_1
BIT_2
...
BIT_15

Return Value: None
Remarks: The bit mask is bitwise AND’d with the contents of the PORTx register. See code example
Source File: None
Code Example: /* read bits 12, 8, 7 of PORTB */
config = mPORTBReadBits(BIT12 | BIT_8 | BIT_7);

mPORTAReadLatch
mPORTBReadLatch
mPORTCReadLatch
mPORTDReadLatch
mPORTEReadLatch
mPORTFReadLatch
mPORTGReadLatch

Description: This macro provides the contents of LATx register.
mPORTAReadLatch
mPORTBReadLatch
mPORTCReadLatch
mPORTDReadLatch
mPORTEReadLatch
mPORTFReadLatch
mPORTGReadLatch (Continued)

Include:    plib.h
Prototype:  
            unsigned int mPORTAReadLatch(void);
            unsigned int mPORTBReadLatch(void);
            unsigned int mPORTCReadLatch(void);
            unsigned int mPORTDReadLatch(void);
            unsigned int mPORTEReadLatch(void);
            unsigned int mPORTFReadLatch(void);
            unsigned int mPORTGReadLatch(void);

Arguments:  None
Return Value:  None
Remarks:     Same as reading the LATx register. See code example
Source File:  None
Code Example: /* read the value in LATA */
              value = mPORTAReadLatch();
mPORTAReadLatchBit
mPORTBReadLatchBit
mPORTCReadLatchBit
mPORTDReadLatchBit
mPORTEREadLatchBit
mPORTFReadLatchBit
mPORTGReadLatchBit

**Description:**
This macro provides the masked contents of LATx register.

**Include:**
plib.h

**Prototype:**

```c
unsigned int mPORTAReadLatchBit(unsigned int _bits);
unsigned int mPORTBReadLatchBit(unsigned int _bits);
unsigned int mPORTCReadLatchBit(unsigned int _bits);
unsigned int mPORTDReadLatchBit(unsigned int _bits);
unsigned int mPORTEREadLatchBit(unsigned int _bits);
unsigned int mPORTFReadLatchBit(unsigned int _bits);
unsigned int mPORTGReadLatchBit(unsigned int _bits);
```

**Arguments:**

```c
_bits
```

This argument contains one or more bit masks bitwise OR'd together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

**IO Pin Bit Masks**

```c
BIT_0
BIT_1
BIT_2
...
BIT_15
```

**Return Value:**
None

**Remarks:**
The bit mask is bitwise AND'd with the contents of the LATx register. See code example

**Source File:**
None

**Code Example:**

```c
/* get the state of bit15 of LATD */
config = mPORTDReadLatchBit(BIT_15);
```
## 10.8 WRITE OPERATIONS

### Macros

<table>
<thead>
<tr>
<th>Macros</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>mPORTAWrite() ...</td>
<td>PORTWrite()</td>
</tr>
<tr>
<td>mPORTGWrite()</td>
<td>PORTSetBits()</td>
</tr>
<tr>
<td>mPORTASetBits() ...</td>
<td>PORTClearBits()</td>
</tr>
<tr>
<td>mPORTGSetBits()</td>
<td>PORTToggleBits()</td>
</tr>
<tr>
<td>mPORTAClearBits() ...</td>
<td></td>
</tr>
<tr>
<td>mPORTGClearBits()</td>
<td></td>
</tr>
<tr>
<td>mPORTAToggleBits() ...</td>
<td></td>
</tr>
<tr>
<td>mPORTGToggleBits()</td>
<td></td>
</tr>
</tbody>
</table>

### Description

Before reading and writing to any I/O port, the data direction of a desired pin must be properly configured as digital input or digital output. Some port I/O pins share digital and analog features and require the analog feature to be disabled when configuring the I/O port pin for digital mode.

### Usage

These functions are typically used early in the program execution to establish the proper mode and state of the general purpose IO pins.

### Advanced

These functions configure port pins as digital input or digital output and automatically disable analog inputs that may be multiplexed with the specified pin(s).

**Feature:** Complete digital I/O pin configuration. These functions meet all the necessary configuration requirements to properly configure port I/O pins that are digital only and those port IO pins that share analog and digital functionality.

**When to use:** These functions provide a simple and preferred method to configure digital I/O pins when the user is not familiar with the details of an I/O port. The user only needs to specify a PORT and PIN(s).

For example:

```
PORTSetPinsDigitalIn(IOPORT_B, BIT_0)
PORTWrite
PORTSetBits
PORTClearBits
PORTToggleBits
```

### Basic

These macros configure port pins as digital input or digital output.

**Feature:** Simple digital I/O pin configuration only.

**When to use:** These macros provide basic digital I/O pin configuration for users who need to control other aspects of the port I/O pin configuration and it is recommended that the user is familiar with the detailed I/O PORT operation. The user is responsible for disabling any analog input that may be multiplexed with the specified pin. The user only needs to specify the PIN(s).

For example:

```
mPORTGToggleBits(BIT_0 | BIT_4)
mPORTxWrite
mPORTxClearBits
mPORTxSetBits
mPORTxToggleBits
```
PORTWrite

Description: This function writes the specified value to the selected PORT register

Include: plib.h

Prototype: void PORTWrite(IO_PORT_ID port, unsigned int bits);

Arguments: port This argument is an IO_PORT_ID which specifies the desired port. Select only one mask from the mask set defined below.

<table>
<thead>
<tr>
<th>IO PORT ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOPORT_A</td>
</tr>
<tr>
<td>IOPORT_B</td>
</tr>
<tr>
<td>IOPORT_C</td>
</tr>
<tr>
<td>IOPORT_D</td>
</tr>
<tr>
<td>IOPORT_E</td>
</tr>
<tr>
<td>IOPORT_F</td>
</tr>
<tr>
<td>IOPORT_G</td>
</tr>
</tbody>
</table>

bits This argument contains one or more bit masks bitwise OR’d together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

<table>
<thead>
<tr>
<th>IO Pin Bit Masks</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT_0</td>
</tr>
<tr>
<td>BIT_1</td>
</tr>
<tr>
<td>BIT_2</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>BIT_15</td>
</tr>
</tbody>
</table>

Return Value: None

Remarks: This function writes directly to the selected PORT register. In this way, all bits in the PORT register are affected.

Source File: port_write_lib.c

Code Example: PORTWrite(IOPORT_B, BIT_5);
or
PORTWrite(IOPORT_B, 0xC4FF);
### PORTSetBits

**Description:** This function sets the selected PORT pins.

**Include:** plib.h

**Prototype:**

```c
void PORTSetBits(IO_PORT_ID port, unsigned int bits);
```

**Arguments:**

- `port` This argument is an IO_PORT_ID which specifies the desired port. Select only one mask from the mask set defined below.

<table>
<thead>
<tr>
<th>IO PORT ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOPORT_A</td>
</tr>
<tr>
<td>IOPORT_B</td>
</tr>
<tr>
<td>IOPORT_C</td>
</tr>
<tr>
<td>IOPORT_D</td>
</tr>
<tr>
<td>IOPORT_E</td>
</tr>
<tr>
<td>IOPORT_F</td>
</tr>
<tr>
<td>IOPORT_G</td>
</tr>
</tbody>
</table>

- `bits` This argument contains one or more bit masks bitwise OR'd together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

<table>
<thead>
<tr>
<th>IO Pin Bit Masks</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT_0</td>
</tr>
<tr>
<td>BIT_1</td>
</tr>
<tr>
<td>BIT_2</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>BIT_15</td>
</tr>
</tbody>
</table>

**Return Value:** None

**Remarks:** This function writes to the corresponding PORTSET register. In this way, only those bits = ‘1’ are SET. All other bits are not affected.

**Source File:** port_set_bits_lib.c

**Code Example:**

```c
PORTSetBits(IOPORT_A, BIT_8 | BIT_7);
```

or

```c
PORTSetBits(IOPORT_F, 0x05);
```
PORTClearBits

Description: This function clears the selected PORT pins.

Include: plib.h

Prototype: void PORTWrite(IO_PORT_ID port, unsigned int bits);

Arguments:

port This argument is an IO_PORT_ID which specifies the desired port. Select only one mask from the mask set defined below.

IO PORT ID
IOPORT_A
IOPORT_B
IOPORT_C
IOPORT_D
IOPORT_E
IOPORT_F
IOPORT_G

bits This argument contains one or more bit masks bitwise OR'd together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

IO Pin Bit Masks
BIT_0
BIT_1
BIT_2
...
BIT_15

Return Value: None

Remarks: This function writes to the corresponding PORTCLR register. In this way, only those bits = ‘1’ are CLEARED. All other bits are not affected.

Source File: port_clear_bits_lib.c

Code Example:
PORTClearBits(IOPORT_C, BIT_2);

or
PORTClearBits(IOPORT_E, 0xFFFF);
PORTToggleBits

**Description:**
This function toggles the selected PORT pins.

**Include:**
plib.h

**Prototype:**
```c
void PORTToggleBits(IO_PORT_ID port, unsigned int bits);
```

**Arguments:**
- **port**: This argument is an IO_PORT_ID which specifies the desired port. Select only one mask from the mask set defined below.

<table>
<thead>
<tr>
<th>IO PORT ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOPORT_A</td>
</tr>
<tr>
<td>IOPORT_B</td>
</tr>
<tr>
<td>IOPORT_C</td>
</tr>
<tr>
<td>IOPORT_D</td>
</tr>
<tr>
<td>IOPORT_E</td>
</tr>
<tr>
<td>IOPORT_F</td>
</tr>
<tr>
<td>IOPORT_G</td>
</tr>
</tbody>
</table>

- **bits**: This argument contains one or more bit masks bitwise OR'd together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

<table>
<thead>
<tr>
<th>IO Pin Bit Masks</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT_0</td>
</tr>
<tr>
<td>BIT_1</td>
</tr>
<tr>
<td>BIT_2</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>BIT_15</td>
</tr>
</tbody>
</table>

**Return Value:**
None

**Remarks:**
This function writes to the corresponding PORTINV register. In this way, only those bits = ‘1’ are TOGGLED. All other bits are not affected.

**Source File:**
port_toggle_bits_lib.c

**Code Example:**
```c
PORTToggleBits(IOPORT_B, BIT_0);
```

or
```c
PORTToggleBits(IOPORT_G, 0x08);
```
### mPORTAWrite

**Description:** This macro writes a value to LATx register.

**Include:** plib.h

**Prototype:**
```c
void mPORTAWrite(unsigned int _value);
```

**Arguments:** `_value`

**Return Value:** None

**Remarks:** See code example

**Source File:** None

**Code Example:**
```c
/* write a value to PORT C */
mPORTCWrite(0x0055);
```
mPORTAClearBits
mPORTBClearBits
mPORTCClearBits
mPORTDClearBits
mPORTEClearBits
mPORTFClearBits
mPORTGClearBits

Description: This macro clears specified IO Port pins.

Include: plib.h

Prototype:
void mPORTAClearBits(unsigned int _bits);
void mPORTBClearBits(unsigned int _bits);
void mPORTCClearBits(unsigned int _bits);
void mPORTDClearBits(unsigned int _bits);
void mPORTEClearBits(unsigned int _bits);
void mPORTFClearBits(unsigned int _bits);
void mPORTGClearBits(unsigned int _bits);

Arguments: _bits This argument contains one or more bit masks bitwise OR’d together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

IO Pin Bit Masks
BIT_0
BIT_1
BIT_2
...
BIT_15

Return Value: None

Remarks: To clear a specific IO pin, include its bit mask in the argument. Argument is copied to the LATCLRx register. If a mask bit is = ‘1’, the corresponding IO pin is driven = 0; if a mask bit = ‘0’, the corresponding IO pin is not affected. See code example

Source File: None

Code Example:
/* Set IO pins PORTA<4,1:0> = 0 */
mPORTAClearBits(BIT_4 | BIT_1 | BIT_0);
mPORTASetBits
mPORTBSetBits
mPORTCSetBits
mPORTDSetBits
mPORTESetBits
mPORTFSetBits
mPORTGSetBits

Description: This macro sets specified IO Port pins.
Include: plib.h
Prototype:
  void mPORTASetBits(unsigned int _bits);
  void mPORTBSetBits(unsigned int _bits);
  void mPORTCSetBits(unsigned int _bits);
  void mPORTDSetBits(unsigned int _bits);
  void PORTESetBits(unsigned int _bits);
  void PORTFSetBits(unsigned int _bits);
  void PORTGSetBits(unsigned int _bits);

Arguments: _bits This argument contains one or more bit masks bitwise OR'd together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

IO Pin Bit Masks
  BIT_0
  BIT_1
  BIT_2
  ...
  BIT_15

Return Value: None

Remarks: To set a specific IO pin, include its bit mask in the argument. Argument is copied to the LATSETx register. If a mask bit is = '1', the corresponding IO pin is driven = '1'; if a mask bit = '0', the corresponding IO pin is not affected. See code example

Source File: None

Code Example:
  /* Set IO pin PORTG<15> = 1 */
  mPORTGSetBits(BIT_15);
mPORTAToggleBits
mPORTBToggleBits
mPORTCToggleBits
mPORTDToggleBits
mPORTEToggleBits
mPORTFToggleBits
mPORTGToggleBits

Description: This macro toggles specified IO Port pins.

Include:

plib.h

Prototype:

void mPORTAToggleBits(unsigned int _bits);
void mPORTBToggleBits(unsigned int _bits);
void mPORTCToggleBits(unsigned int _bits);
void mPORTDToggleBits(unsigned int _bits);
void mPORTEToggleBits(unsigned int _bits);
void mPORTFToggleBits(unsigned int _bits);
void mPORTGToggleBits(unsigned int _bits);

Arguments: _bits

This argument contains one or more bit masks bitwise OR’d together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes corresponding bit(s) are disabled, or default value, and will be set = 0.

IO Pin Bit Masks
BIT_0
BIT_1
BIT_2
...
BIT_15

Return Value: None

Remarks: To toggle a specific IO pin, include its bit mask in the argument. Argument is copied to the LATINVx register. If a mask bit is ‘1’, the corresponding IO pin is toggles the current state of the IO pin; if a mask bit = ‘0’, the corresponding IO pin is not affected. See code example

Source File: None

Code Example: /* Toggle PORTB<2:1> */

mPORTBToggleBits(BIT_2 | BIT_1);
10.9 MISC OPERATIONS

<table>
<thead>
<tr>
<th>Macros</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>mJTAGPortEnable( )</td>
<td>PORTResetPins( )</td>
</tr>
</tbody>
</table>

- **Description**
  Before reading and writing to any I/O port, the data direction of a desired pin must be properly configured as digital input or digital output. Some port I/O pins share digital and analog features and require the analog feature to be disabled when configuring the I/O port pin for digital mode.

- **Usage**
  These functions are typically used early in the program execution to establish the proper mode and state of the general purpose IO pins.

- **Advanced**
  These functions configure port pins as digital input or digital output and automatically disable analog inputs that may be multiplexed with the specified pin(s).
  
  **Feature:** Complete digital IO pin configuration. These functions meet all the necessary configuration requirements to properly configure port IO pins that are digital only and those port IO pins that share analog and digital functionality.

  **When to use:** These functions provide a simple and preferred method to configure digital I/O pins when the user is not familiar with the details of an I/O port. The user only needs to specify a PORT and PIN(s).

  For example: PORTResetPins(IOPORT_B, BIT_0)

- **Basic**
  These macros configure port pins as digital input or digital output.

  **Feature:** Simple digital IO pin configuration only.

  **When to use:** These macros provide basic digital IO pin configuration for users who need to control other aspects of the port IO pin configuration and it is recommended that the user is familiar with the detailed IO PORT operation. The user is responsible for disabling any analog input that may be multiplexed with the specified pin. The user only needs to specify the PIN(s).

  For example: mJTAGPortEnable(DEBUG_JTAGPORT_OFF)

  mJTAGPortEnable()
mJTAGPortEnable (Continued)

mJTAGPortEnable()
PORTResetPins()

Return value: None
Remarks: See code example.
Source File: None
Code Example: /* disable the JTAG Port */
mJTAGPortEnable(0);

PORTResetPins

Description: This function sets the specified pins to their reset state.
Include: plib.h
Prototype: void PORTResetPins(IoPortId portId, unsigned int _bits);

Arguments: port This argument is an IO_PORT_ID which specifies the
desired port. Select only one mask from the mask set defined below.

IO PORT ID
IOPORT_A
IOPORT_B
IOPORT_C
IOPORT_D
IOPORT_E
IOPORT_F
IOPORT_G

_bits This argument contains one or more bit masks bitwise OR'd
together. Select one or more masks from the mask set defined below. Note: An absent mask symbol assumes
.corresponding bit(s) are disabled, or default value, and
will be set = 0.

IO Pin Bit Masks
BIT_0
BIT_1
BIT_2
...
BIT_15

Return Value: None
Remarks: See code example.
Source File: port_reset_pins.c
Code Example: /* Reset port pins */
PORTResetPins(IOPORT_A, BIT_0);
10.10 Example of Use

```
#define CONFIG       (CN_ON  | CN_IDLE_CON)
#define PINS         (CN15_ENABLE)
#define PULLUPS      (CN_PULLUP_DISABLE_ALL)
#define INTERRUPT    (CHANGE_INT_ON | CHANGE_INT_PRI_2)

void delay(unsigned int);

int main(void)
{
    unsigned short value;
    // STEP 1. configure the wait states and peripheral bus clock
    SYSTEMConfigWaitStatesAndPB(72000000L);
    // STEP 2. configure the port registers
    PORTSetPinsDigitalOut(IOPORT_A, BIT_2 | BIT_3);
    PORTSetPinsDigitalIn(IOPORT_D, BIT_6);
    // STEP 3. initialize the port pin states = outputs low
    mPORTAClearBits(BIT_2 | BIT_3);
    // STEP 4. setup the change notice options
    mCNOpen(CONFIG, PINS, PULLUPS);
    // STEP 5. read port(s) to clear mismatch on change notice pins
    value = mPORTDRead();
    // STEP 6. clear change notice interrupt flag
    ConfigIntCN(INTERRUPT);
    // STEP 7. enable multi-vector interrupts
    INTEnableSystemMultiVectoredInt();
    while(1)
    {
        mPORTASetBits(BIT_2);        // BIT_2 = 1
        delay(10E3);
        mPORTAClearBits(BIT_2);      // BIT_2 = 0
        delay(10E3);
    }
}

void __ISR(_CHANGE_NOTICE_VECTOR, ipl2) ChangeNotice_Handler(void)
{
    // clear the mismatch condition
    mPORTDRead();
    // clear the interrupt flag
    mCNClearIntFlag();
    // toggle the led
    mPORTAToggleBits(BIT_3);       // BIT_3 = TOGGLE
    asm("nop");
    // ... things to do .. add code here
}
void delay(unsigned int count)
{
    while(--count);
}
```
11.0 TIMER FUNCTIONS

The PIC32MX TIMER library consists of functions and macros supporting common configuration and control features.

• CPU Core Timer Operations
  OpenCoreTimer
  UpdateCoreTimer
  mConfigIntCoreTimer
  mEnableIntCoreTimer
  mDisableIntCoreTimer
  mSetPriorityIntCoreTimer
  ReadCoreTimer
  WriteCoreTimer

• General Purpose Timer Common Operations
  OpenTimerx
  CloseTimerx
  ConfigIntTimerx
  SetPriorityIntTx
  DisableIntTx
  EnableIntTx

• General Purpose Timer and Period Read/Write Operations
  ReadTimerx
  WriteTimerx
  ReadPeriodx
  WritePeriodx
## 11.1 CPU Core Timer Functions and Macros

### OpenCoreTimer

**Description:** This function configures the 32-bit CPU Core Timer registers.

**Include:** 
plib.h

**Prototype:**

```c
void OpenCoreTimer(unsigned int compare);
```

**Arguments:**

- `period` This argument contains a 32-bit period value for the CPU Core Compare register.

**Return Value:** None

**Remarks:**

This function clears the CPU Core Count register, then loads the CPU Core Compare register with `period`.

**Source File:**

- OpenCoreTimer(0x00004000);

### UpdateCoreTimer

**Description:** This function updates the 32-bit CPU Core Compare register.

**Include:** 
plib.h

**Prototype:**

```c
void UpdateCoreTimer(unsigned int period);
```

**Arguments:**

- `period` This argument contains a 32-bit period value for the CPU Core Compare register.

**Return Value:** None

**Remarks:**

This function adds `period` to the current value in the CPU Core Compare register, effectively creating the next period match.

**Note:** A simple method for creating periodic interrupts can be achieved by using the CPU Core Timer and an ISR (Interrupt Service Routine) that calls `UpdateCoreTimer()` to update the Core Compare value.

See Core Timer code example at the end of this chapter.

**Source File:**

**Code Example:**

```c
void _CoreTimerHandler(void)
{
    mCTClearIntFlag();
    UpdateCoreTimer(CORE_TIMER_PERIOD);

    // .. things to do .. add code here
}
```
mConfigIntCoreTimer

Description: This function configures the 32-bit CPU Core Timer interrupt.
Include: plib.h
Prototype: void mConfigIntCoreTimer(config);
Arguments: config This argument contains one or more bit masks bitwise OR'd together. Select one or more from the following bit masks. Note: An absent mask symbol in an argument assumes the corresponding bit(s) are disabled, or default value, and are set = 0.

Core Timer On/Off
  CT_INT_ON
  CT_INT_OFF

Core Timer Priority Interrupt Level
  CT_INT_PRIOR_7
  CT_INT_PRIOR_6
  CT_INT_PRIOR_5
  CT_INT_PRIOR_4
  CT_INT_PRIOR_3
  CT_INT_PRIOR_2
  CT_INT_PRIOR_1
  CT_INT_PRIOR_0

Core Timer Sub-Priority Interrupt Level
  CT_INT_SUB_PRIOR_3
  CT_INT_SUB_PRIOR_2
  CT_INT_SUB_PRIOR_1
  CT_INT_SUB_PRIOR_0

Return Value: None
Remarks: This macro clears the Core Timer interrupt flag, sets the priority and sub-priority interrupt level then enables the Core Timer interrupt.
Source File: 
Code Example: mConfigIntCoreTimer(CT_INT_ON | CT_INT_PRIOR_4);

mEnableIntCoreTimer

Description: This macro enables the 32-bit CPU Core Timer interrupt.
Include: plib.h
Prototype: mEnableIntCoreTimer();
Arguments: None
Return Value: None
Remarks: 
Source File: 
Code Example: mEnableIntCoreTimer();
### mDisableIntCoreTimer

**Description:** This macro disables the 32-bit CPU Core Timer interrupt.

**Include:** plib.h

**Prototype:**
```c
void mDisableIntCoreTimer(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:**

**Source File:**

**Code Example:**
```c
mDisableIntCoreTimer();
```

### mSetPriorityIntCoreTimer

**Description:** This macro sets the 32-bit CPU Core Timer interrupt priority.

**Include:** plib.h

**Prototype:**
```c
void mCTSetIntPriority(unsigned int priority);
```

**Arguments:**

- `priority` This argument is the priority value for the CPU Core Timer interrupt level.

  **Core Timer Priority Interrupt Levels**
  - CT_INT_PRIOR_7
  - CT_INT_PRIOR_6
  - CT_INT_PRIOR_5
  - CT_INT_PRIOR_4
  - CT_INT_PRIOR_3
  - CT_INT_PRIOR_2
  - CT_INT_PRIOR_1
  - CT_INT_PRIOR_0

**Return Value:** None

**Remarks:** This function modifies the previously set priority without any need to specify other parameters.

**Source File:**

**Code Example:**
```c
mCTSetIntPriority(CT_INT_PRIOR_2);
```
**ReadCoreTimer**

**Description:** This function returns the 32-bit CPU Core Timer register value.

**Include:** plib.h

**Prototype:**

```c
unsigned int ReadCoreTimer(void);
```

**Arguments:** None

**Return Value:** 32-bit Core Timer value.

**Remarks:**

**Source File:**

**Code Example:**

```c
unsigned int t0;
t0 = ReadCoreTimer();
```

---

**WriteCoreTimer**

**Description:** This function writes a 32-bit value to the CPU Core Timer register.

**Include:** plib.h

**Prototype:**

```c
void WriteCoreTimer(unsigned int timer);
```

**Arguments:**

- `period` This argument is the 32-bit period value written to the CPU Core Timer register.

**Return Value:** None

**Remarks:** This function writes value `timer` to the Core Timer register.

**Source File:**

**Code Example:**

```c
WriteCoreTimer(0x12345678);
```
11.2 General Purpose Timer Functions and Macros

OpenTimer1
OpenTimer2
OpenTimer3
OpenTimer4
OpenTimer5

| Description: | This macro configures the 16-bit timer module. |
| Include: | plib.h |
| Prototype: | void OpenTimer1(unsigned int config, unsigned int period);
void OpenTimer2(unsigned int config, unsigned int period);
void OpenTimer3(unsigned int config, unsigned int period);
void OpenTimer4(unsigned int config, unsigned int period);
void OpenTimer5(unsigned int config, unsigned int period);

| Arguments: | config |
| | This argument contains one or more bit masks bitwise OR'd together. Select one or more from the following bit mask. Note: An absent mask symbol in an argument assumes the corresponding bit(s) are disabled, or default value, and are set = 0. |
| | Timer Module On/Off |
| | Tx_ON |
| | Tx_OFF |
| | (These bit fields are mutually exclusive) |
| | Asynchronous Timer Write Disable |
| | T1_TMWDIS_ON |
| | T1_TMWDIS_OFF |
| | (These bit field(s) are mutually exclusive) |
| | Timer Module Idle mode On/Off |
| | Tx_IDLE_CON |
| | Tx_IDLE_STOP |
| | (These bit field(s) are mutually exclusive) |
| | Timer Gate time accumulation enable |
| | Tx_GATE_ON |
| | Tx_GATE_OFF |
| | (These bit field(s) are mutually exclusive) |
### OpenTimer1

<table>
<thead>
<tr>
<th>Timer Prescaler(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1_PS_1_1</td>
</tr>
<tr>
<td>T1_PS_1_8</td>
</tr>
<tr>
<td>T1_PS_1_64</td>
</tr>
<tr>
<td>T1_PS_1_256</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Timer Prescaler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tx_PS_1_1</td>
</tr>
<tr>
<td>Tx_PS_1_2</td>
</tr>
<tr>
<td>Tx_PS_1_4</td>
</tr>
<tr>
<td>Tx_PS_1_8</td>
</tr>
<tr>
<td>Tx_PS_1_16</td>
</tr>
<tr>
<td>Tx_PS_1_32</td>
</tr>
<tr>
<td>Tx_PS_1_64</td>
</tr>
<tr>
<td>Tx_PS_1_256</td>
</tr>
</tbody>
</table>

(These bit fields are mutually exclusive)

<table>
<thead>
<tr>
<th>Timer Synchronous clock enable(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tx_SYNC_EXT_ON</td>
</tr>
</tbody>
</table>

(These bit fields are mutually exclusive)

<table>
<thead>
<tr>
<th>Timer Clock source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tx_SOURCE_EXT</td>
</tr>
</tbody>
</table>

(These bit fields are mutually exclusive)

### Description:

This function configures Timer2 and Timer3 pair or Timer4 and Timer5 pair as 32-bit timers.

### Include:

plib.h

### Prototype:

```c
void OpenTimer32(unsigned int config, 
                  unsigned long period);
void OpenTimer45(unsigned int config, 
                  unsigned long period);
```

### Remarks:

This macro clears the TMRx register, writes period to the PRx register and writes config to the TxCON register.

### Code Example:

```c
/* Enable timer1; external clock source; synchronized timer; prescaler 1:8; load 0xFFFF in period register PR1 */

OpenTimer1(T1_ON | T1_SOURCE_EXT | T1_SYNC_EXT_ON | T1_PS_1_8, 0xFFFF);
```

**Note 1:** Use with Timer1 only
OpenTimer23
OpenTimer45 (Continued)

Arguments:  config

This argument contains one or more bit masks bitwise OR'd together. Select one or more from the following bit mask.

Note: An absent mask symbol in an argument assumes the corresponding bit(s) are disabled, or default value, and are set = 0.

Note: Replace bits masks using 'x' with '2' for OpenTimer23, '4' for OpenTimer45.

Timer module On/Off
Tx_ON
Tx_OFF
(These bit fields are mutually exclusive)

Timer Module Idle mode On/Off
Tx_IDLE_CON
Tx_IDLE_STOP
(These bit fields are mutually exclusive)

Timer Gate time accumulation enable
Tx_GATE_ON
Tx_GATE_OFF
(These bit fields are mutually exclusive)

Timer prescaler
Tx_PS_1_1
Tx_PS_1_2
Tx_PS_1_4
Tx_PS_1_8
Tx_PS_1_16
Tx_PS_1_32
Tx_PS_1_64
Tx_PS_1_256
(These bit fields are mutually exclusive)

32-bit Timer Mode enable
Tx_32BIT_MODE_ON
Tx_32BIT_MODE_OFF
(These bit fields are mutually exclusive)

Timer clock source
Tx_SOURCE_EXT
Tx_SOURCE_INT
(These bit fields are mutually exclusive)

period
This contains the period match value to be stored into the 32-bit PR register.

Return Value: None

Remarks: OpenTimer23() clears the TMR23 register pair, writes period to the PR23 register pair and writes config to the T2CON register.

Note: This macro also sets the T2CON<T32> bit = 1.

OpenTimer45() clears TMR45 register pair, writes period to the PR45 register pair and writes config to the T4CON register.

Note: This macro also sets the T4CON<T32> bit = 1.

Source File:
OpenTimer23

OpenTimer45  (Continued)

Code Example:
/* Enable timer pair timer2/timer3; prescaler 1:256;
set 0x00A00000 as the period */

OpenTimer23(T2_ON | T2_PS_1_256 | T2_32BIT_MODE_ON, 0x00A00000);

CloseTimer1
CloseTimer2
CloseTimer3
CloseTimer4
CloseTimer5

Description:  This macro turns off the 16-bit timer module.
Include:  plib.h
Prototype:  void CloseTimer1(void);
void CloseTimer2(void);
void CloseTimer3(void);
void CloseTimer4(void);
void CloseTimer5(void);

Arguments:  None
Return Value:  None
Remarks:  CloseTimer() disables clears the appropriate TxlE interrupt enable bit
and clears all bits in the  TxCON register.

Source File:  Code Example:  CloseTimer1();

Close23Timer
Close45Timer

Description:  This macro turns off the 32-bit timer module.
Include:  plib.h
Prototype:  void CloseTimer23 (void)
void CloseTimer45 (void)

Arguments:  None
Return Value:  None
Remarks:  CloseTimer23() calls CloseTimer2() and Close Timer3().
CloseTimer45() calls CloseTimer4() and Close Timer5().

Source File:  Code Example:  CloseTimer23();
**Description:**
This macro configures the 16-bit timer interrupt.

**Include:**
plib.h

**Prototype:**

```c
void ConfigIntTimer1(unsigned int config);
void ConfigIntTimer2(unsigned int config);
void ConfigIntTimer3(unsigned int config);
void ConfigIntTimer4(unsigned int config);
void ConfigIntTimer5(unsigned int config);
```

**Arguments:**
config
This argument contains one or more bit masks bitwise OR’d together. Select one or more from the following bit mask.

**Note:** An absent mask symbol in an argument assumes the corresponding bit(s) are disabled, or default value, and are set = 0.

- **Timer interrupt enable/disable**
  - Tx_INT_ON
  - Tx_INT_OFF
  (These bit fields are mutually exclusive)

- **Timer interrupt priorities**
  - Tx_INT_PRIOR_7
  - Tx_INT_PRIOR_6
  - Tx_INT_PRIOR_5
  - Tx_INT_PRIOR_4
  - Tx_INT_PRIOR_3
  - Tx_INT_PRIOR_2
  - Tx_INT_PRIOR_1
  - Tx_INT_PRIOR_0
  (These bit fields are mutually exclusive)

- **Timer interrupt sub-priorities**
  - Tx_INT_SUB_PRIOR_3
  - Tx_INT_SUB_PRIOR_2
  - Tx_INT_SUB_PRIOR_1
  - Tx_INT_SUB_PRIOR_0
  (These bit fields are mutually exclusive)

**Return Value:**
None

**Remarks:**
This macro configures the Timer interrupt.

**Source File:**

**Code Example:**

```c
/* Timer 1; Enable Timer, & set priority level 2 */
ConfigIntTimer1(T1_INT_ON | T1_INT_PRIOR_2);
```
ConfigIntTimer23
ConfigIntTimer45

Description: This macro configures the 32-bit timer interrupt.
Include: plib.h
Prototype: void ConfigIntTimer23(unsigned int config);
          void ConfigIntTimer45(unsigned int config);

Arguments: config This argument contains one or more bit masks bitwise OR'd together. Select one or more from the following bit mask.
            Note: An absent mask symbol in an argument assumes the corresponding bit(s) are disabled, or default value, and are set = 0.

            Note: use the following replacements for 'x':
            23 for ConfigIntTimer23();
            45 for ConfigIntTimer45().

            Timer interrupt enable/disable
            T_x_INT_ON
            T_x_INT_OFF
            (These bit fields are mutually exclusive)

            Timer interrupt priorities
            T_x_INT_PRIOR_7
            T_x_INT_PRIOR_6
            T_x_INT_PRIOR_5
            T_x_INT_PRIOR_4
            T_x_INT_PRIOR_3
            T_x_INT_PRIOR_2
            T_x_INT_PRIOR_1
            T_x_INT_PRIOR_0
            (These bit fields are mutually exclusive)

            Timer interrupt sub- priorities
            T_x_INT_SUB_PRIOR_3
            T_x_INT_SUB_PRIOR_2
            T_x_INT_SUB_PRIOR_1
            T_x_INT_SUB_PRIOR_0
            (These bit fields are mutually exclusive)

Return Value: None
Remarks: ConfigIntTimer23() configures Timer3 interrupt.
          ConfigIntTimer45() configures Timer5 interrupt.

Source File: 
Code Example: /* Set Timer45 interrupt priority = 3, sub = 2 */

            ConfigIntTimer45(T45_INT_ON | T45_INT_PRIOR_3 | T45_INT_SUB_PRIOR_2);
### Description:
This macro configures the timer interrupt priority.

### Include:
plib.h

### Prototype:
```c
void SetPriorityIntT1(unsigned int config);
void SetPriorityIntT2(unsigned int config);
void SetPriorityIntT3(unsigned int config);
void SetPriorityIntT4(unsigned int config);
void SetPriorityIntT5(unsigned int config);
```

### Arguments:

- **config**
  This argument contains one or more bit masks bitwise OR'd together. Select one or more from the following bit mask.
  Note: An absent mask symbol in an argument assumes the corresponding bit(s) are disabled, or default value, and are set = 0.

  **Timer interrupt priorities**
  - `Tx_INT_PRIOR_7`
  - `Tx_INT_PRIOR_6`
  - `Tx_INT_PRIOR_5`
  - `Tx_INT_PRIOR_4`
  - `Tx_INT_PRIOR_3`
  - `Tx_INT_PRIOR_2`
  - `Tx_INT_PRIOR_1`
  - `Tx_INT_PRIOR_0`

  (These bit fields are mutually exclusive)

### Return Value:
None

### Remarks:
This macro configures the appropriate TxIP interrupt priority bits.

### Source File:
Code Example:

```c
/* Set Timer3 interrupt priority = 2*/
SetPriorityIntT3(T3_INT_PRIOR_2);
```
SetPriorityIntT23
SetPriorityIntT45

| Description: | This macro configures the a timer's interrupt priority. |
| Include: | plib.h |
| Prototype: | void SetPriorityIntT23(unsigned int config);
void SetPriorityIntT45(unsigned int config); |

Arguments: | config This argument contains one or more bit masks bitwise OR'd together. Select one or more from the following bit mask. Note: An absent mask symbol in an argument assumes the corresponding bit(s) are disabled, or default value, and are set = 0.
Note: use the following replacements for 'x':
23 for SetPriorityIntT23();
45 for SetPriorityIntT45().

Timer interrupt priorities
Tx_INT_PRIOR_7
Tx_INT_PRIOR_6
Tx_INT_PRIOR_5
Tx_INT_PRIOR_4
Tx_INT_PRIOR_3
Tx_INT_PRIOR_2
Tx_INT_PRIOR_1
Tx_INT_PRIOR_0
(These bit fields are mutually exclusive)

Return Value: | None |
Remarks: | SetPriorityIntT23() configures Timer3 interrupt.
SetPriorityIntT45() configures Timer5 interrupt. |

Source File: |
Code Example: | /* Set Timer23 interrupt priority = 2*/
SetPriorityIntT23(T23_INT_PRIOR_2); |
DisableIntT1
DisableIntT2
DisableIntT3
DisableIntT4
DisableIntT5

Description: This macro disables the a timer's interrupt.
Include: plib.h
Prototype: void DisableIntT1(void);
void DisableIntT2(void);
void DisableIntT3(void);
void DisableIntT4(void);
void DisableIntT5(void);

Arguments: None
Return Value: None
Remarks: This macro clears the appropriate TxIE interrupt enable bit.
Source File: Code Example:
/* Disable Timer4 interrupt */
DisableIntT4();

DisableIntT23
DisableIntT45

Description: This macro disables the a timer's interrupt.
Include: plib.h
Prototype: void DisableIntT23(void);
void DisableIntT45(void);

Arguments: None
Return Value: None
Remarks: DisableIntT23() clears the T3IE interrupt enable bit.
DisableIntT45() clears the T5IE interrupt enable bit.
Source File: Code Example:
/* Disable Timer45 interrupt */
DisableIntT45();
**EnableIntT1**  
**EnableIntT2**  
**EnableIntT3**  
**EnableIntT4**  
**EnableIntT5**

| Description: | This macro enables the a timer's interrupt. |
| Include:     | plib.h |
| Prototype:   | Void EnableIntT1(void);  
               | Void EnableIntT2(void);  
               | Void EnableIntT3(void);  
               | Void EnableIntT4(void);  
               | Void EnableIntT5(void); |

| Arguments:   | None |
| Return Value:| None |
| Remarks:     | This macro sets the appropriate TxIE interrupt enable bit. |

**Source File:**  
**Code Example:**  
/* Enable Timer4 interrupt */  
EnableIntT4();

---

**EnableIntT23**  
**EnableIntT45**

| Description: | This macro enables the a timer's interrupt. |
| Include:     | plib.h |
| Prototype:   | Void EnableIntT23(void);  
               | Void EnableIntT45(void); |

| Arguments:   | None |
| Return Value:| None |
| Remarks:     | EnableIntT23() sets the T3IE interrupt enable bit.  
               | EnableIntT45() sets the T5IE interrupt enable bit. |

**Source File:**  
**Code Example:**  
/* Enable Timer45 interrupt */  
EnableIntT45();
## 11.3 Timer Read/Write Functions and Macros

### ReadTimer1
#### Description:
This macro returns 16-bit timer value.

#### Include:
plib.h

#### Prototype:
```c
unsigned int ReadTimer1(void);
```

#### Arguments:
None

#### Return Value:
16-bit timer

#### Remarks:
This macro returns the contents of the 16-bit timer module timer register.

#### Source File:

#### Code Example:
```c
/* Read timer 4 */
currentValue = ReadTimer4();
```

### ReadTimer2
#### Description:
This macro returns 16-bit timer value.

#### Include:
plib.h

#### Prototype:
```c
unsigned int ReadTimer2(void);
```

### ReadTimer3
#### Description:
This macro returns 16-bit timer value.

#### Include:
plib.h

#### Prototype:
```c
unsigned int ReadTimer3(void);
```

### ReadTimer4
#### Description:
This macro returns 16-bit timer value.

#### Include:
plib.h

#### Prototype:
```c
unsigned int ReadTimer4(void);
```

### ReadTimer5
#### Description:
This macro returns 16-bit timer value.

#### Include:
plib.h

#### Prototype:
```c
unsigned int ReadTimer5(void);
```

#### Arguments:
None

#### Return Value:
16-bit timer

#### Remarks:
This macro returns the contents of the 16-bit timer module timer register.

#### Source File:

#### Code Example:
```c
/* Read timer 4 */
currentValue = ReadTimer4();
```

### ReadTimer23
#### Description:
This function returns 32-bit timer value.

#### Include:
plib.h

#### Prototype:
```c
unsigned int ReadTimer23(void);
```

### ReadTimer45
#### Description:
This function returns 32-bit timer value.

#### Include:
plib.h

#### Prototype:
```c
unsigned int ReadTimer45(void);
```
### WriteTimer1

**Description:**
This function writes a 16-bit timer value.

**Include:**
plib.h

**Prototype:**
void WriteTimer1(unsigned int);
void WriteTimer2(unsigned int);
void WriteTimer3(unsigned int);
void WriteTimer4(unsigned int);
void WriteTimer5(unsigned int);

**Arguments:**
16-bit timer value

**Return Value:**
None

**Remarks:**
This function loads given Timer with the value.

**Source File:**

**Code Example:**
```c
/* Write timer 1 */

void WriteTimer1(unsigned int value);

WriteTimer1(0x0400);
```

### WriteTimer23

**Description:**
This macro writes a 32-bit Timer value.

**Include:**
plib.h

**Prototype:**
void WriteTimer23(unsigned int);
void WriteTimer45(unsigned int);

**Arguments:**
32-bit timer value

**Return Value:**
None

**Remarks:**
This macro writes the 32-bit value into the TMR register pair.

**Source File:**

**Code Example:**
```c
/* Write timer 45 */

void WriteTimer45(unsigned int value);

WriteTimer45(0x00000000);
```
### 11.4 Period Read/Write Functions and Macros

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Include</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadPeriod1</td>
<td>This macro returns 16-bit Period value.</td>
<td>plib.h</td>
<td>unsigned int ReadPeriod1(void);</td>
</tr>
<tr>
<td>ReadPeriod2</td>
<td></td>
<td></td>
<td>unsigned int ReadPeriod2(void);</td>
</tr>
<tr>
<td>ReadPeriod3</td>
<td></td>
<td></td>
<td>unsigned int ReadPeriod3(void);</td>
</tr>
<tr>
<td>ReadPeriod4</td>
<td></td>
<td></td>
<td>unsigned int ReadPeriod4(void);</td>
</tr>
<tr>
<td>ReadPeriod5</td>
<td></td>
<td></td>
<td>unsigned int ReadPeriod5(void);</td>
</tr>
<tr>
<td>ReadPeriod23</td>
<td>This macro returns 32-bit Period value.</td>
<td>plib.h</td>
<td>unsigned int ReadPeriod23(void);</td>
</tr>
<tr>
<td>ReadPeriod45</td>
<td></td>
<td></td>
<td>unsigned int ReadPeriod45(void);</td>
</tr>
</tbody>
</table>

**Arguments:** None  
**Return Value:** 16-bit Period  
**Remarks:** This macro returns the contents of the 16-bit PR register.

**Source File:**  
**Code Example:**  
```c
/* Read Period 4 */
currentValue = ReadPeriod4();
```

**Arguments:** None  
**Return Value:** 32-bit Period  
**Remarks:** This function returns the contents of the 32-bit PR register pair  
**Source File:**  
**Code Example:**  
```c
/* Read Period 45 */
currentValue = ReadPeriod45();
```
### WritePeriod1
#### Description:
This macro writes a 16-bit Period value.

#### Include:
plib.h

#### Prototype:
- void WritePeriod1(unsigned int);
- void WritePeriod2(unsigned int);
- void WritePeriod3(unsigned int);
- void WritePeriod4(unsigned int);
- void WritePeriod5(unsigned int);

#### Arguments:
16-bit Period value

#### Return Value:
None

#### Remarks:
This function loads Period register with the value.

#### Source File:

#### Code Example:
```c
/* Write Period 1 */

WritePeriod1(0x0400);
```

### WritePeriod23
#### Description:
This macro writes a 32-bit Period value.

#### Include:
plib.h

#### Prototype:
- void WritePeriod23(unsigned int);
- void WritePeriod45(unsigned int);

#### Arguments:
32-bit Period value

#### Return Value:
None

#### Remarks:
This macro writes the 32-bit value into the 32-bit Period register.

#### Source File:

#### Code Example:
```c
/* Write Period 45 */

WritePeriod45(0x00000000);
```
11.5 Example: Using Core Timer to generate periodic interrupt

The following code example illustrates the PIC32MX CPU Core Timer and ISR (Interrupt Service Routine) generating a 10 msec (100 tick / second) periodic interrupt.

Note: The PIC32MX CPU Core Timer Compare register must be updated with a new period match value after each match occurs. See function UpdateCoreTimer().

A typical application is a kernel time tick for RTOS or simple scheduler.

```c
#include <plib.h>
/* This example assumes the CPU Core is operating at 60MHz */
#define FOSC 60E6
#define CORE_TICK_PERIOD (FOSC/100)

int main(void)
{
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // STEP 1. configure the core timer
    OpenCoreTimer(CORE_TICK_PERIOD);
    
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // STEP 2. set core timer interrupt level = 2
    mConfigIntCoreTimer(CT_INT_ON | CT_INT_PRIOR_2);
    
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // STEP 3. enable multi-vector interrupts
    INTEnableSystemMultiVectoredInt();
    
    ... do something useful here ...
    
    while(1);
}

/* Core Timer ISR */
/* Specify Interrupt Priority Level = 2, Vector 0 */

void __ISR(_CORE_TIMER_VECTOR, ipl2) _CoreTimerHandler(void)
{
    // clear the interrupt flag
    mCTClearIntFlag();
    
    // update the period
    UpdateCoreTimer(CORE_TICK_PERIOD);
    
    // .. things to do ..
}
```
11.6 Code Example: Using Timer 1 to generate periodic interrupt

The following code example illustrates a 16-bit Timer and ISR (Interrupt Service Routine) generating a 250 msec (4 tick / second) periodic interrupt.

Note: The PIC32MX peripheral timers do not require the period match value be reloaded after each match occurs.

```c
#include <p32xxxx.h>
#include <plib.h>

/* This example assumes the CPU Core is operating at 60MHz */

#define FOSC 60E6
#define PB_DIV 8
#define PRESCALE 256
#define T1_TICK (FOSC/PB_DIV/PRESCALE/4)

int main(void)
{

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // STEP 1. configure the Timer1
    OpenTimer1(T1_ON | T1_SOURCE_INT | T1_PS_1_256, T1_TICK);
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // STEP 2. set the timer interrupt to priority level 2
    ConfigIntTimer1(T1_INT_ON | T1_INT_PRIOR_2);
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // STEP 3. enable multi-vector interrupts
    INTEnableSystemMultiVectoredInt();

    ... do something useful here ...

    while(1);
}
/* Timer 1 ISR */
/* Specify Interrupt Priority Level = 2, Vector 4 */

void __ISR(TIMER_1_INT_VECTOR, ipl2) _Timer1Handler(void)
{
    // clear the interrupt flag
    mT1ClearIntFlag();

    // . things to do ..
```
12.0 INPUT CAPTURE FUNCTIONS

This section contains a list of individual functions for Input Capture module and an example of use of the functions. Functions may be implemented as macros.

12.1 Individual Functions and Macros

OpenCapture1
OpenCapture2
OpenCapture3
OpenCapture4
OpenCapture5

Description: This function configures the Input Capture module.

Include: plib.h

Prototype:

void OpenCapture1(unsigned int config);
void OpenCapture2(unsigned int config);
void OpenCapture3(unsigned int config);
void OpenCapture4(unsigned int config);
void OpenCapture5(unsigned int config);

Arguments:
config This contains the parameters to be configured in the ICxCON register as defined below:

On/Off Control
IC_ON
IC_OFF
(These bit fields are mutually exclusive)

Idle mode operation
IC_IDLE_CON
IC_IDLE_STOP
(These bit fields are mutually exclusive)

First Edge
IC_FEDGE_RISE
IC_FEDGE_FALL
(These bit fields are mutually exclusive)

32 Bit Mode
IC_CAP_32BIT
IC_CAP_16BIT (These bit fields are mutually exclusive)

Timer select
IC_TIMER2_SRC
IC_TIMER3_SRC
(These bit fields are mutually exclusive)

Captures per interrupt
IC_INT_4CAPTURE
IC_INT_3CAPTURE
IC_INT_2CAPTURE
IC_INT_1CAPTURE
(These bit fields are mutually exclusive)
OpenCapture1 (Continued)
OpenCapture2
OpenCapture3
OpenCapture4
OpenCapture5

<table>
<thead>
<tr>
<th>IC mode select</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC_INTERRUPT</td>
</tr>
<tr>
<td>IC_SP_EVERY_EDGE</td>
</tr>
<tr>
<td>IC_EVERY_16_RISE_EDGE</td>
</tr>
<tr>
<td>IC_EVERY_4_RISE_EDGE</td>
</tr>
<tr>
<td>IC_EVERY_RISE_EDGE</td>
</tr>
<tr>
<td>IC_EVERY_FALL_EDGE</td>
</tr>
<tr>
<td>IC_EVERY_EDGE</td>
</tr>
<tr>
<td>IC_INPUTCAP_OFF</td>
</tr>
</tbody>
</table>

(These bit fields are mutually exclusive)

Return Value: None
Remarks: This function configures the Input Capture Module Control register (ICxCON).
Code Example: OpenCapture1(IC_IDLE_CON & IC_TIMER2_SRC & IC_INT_1CAPTURE & IC_EVERY_RISE_EDGE);

CloseCapture1
CloseCapture2
CloseCapture3
CloseCapture4
CloseCapture5

Description: This function turns off the Input Capture module.
Include: plib.h
Prototype:

void CloseCapture1(void);
void CloseCapture2(void);
void CloseCapture3(void);
void CloseCapture4(void);
void CloseCapture5(void);

Arguments: None
Return Value: None
Remarks: This function disables the Input Capture interrupt and then turns off the module. The Interrupt Flag bit is also cleared.
Code Example: CloseCapture1();
ConfigIntCapture1
ConfigIntCapture2
ConfigIntCapture3
ConfigIntCapture4
ConfigIntCapture5
ConfigIntCapture6
ConfigIntCapture7
ConfigIntCapture8

Description: This function configures the Input Capture interrupt.
Include: plib.h
Prototype:
void ConfigIntCapture1(unsigned int config);
void ConfigIntCapture2(unsigned int config);
void ConfigIntCapture3(unsigned int config);
void ConfigIntCapture4(unsigned int config);
void ConfigIntCapture5(unsigned int config);

Arguments: config Input Capture interrupt priority and enable/disable information as defined below:

Interrupt enable/disable
IC_INT_ON
IC_INT_OFF
(These bit fields are mutually exclusive)

Interrupt Priority
IC_INT_PRIOR_0
IC_INT_PRIOR_1
IC_INT_PRIOR_2
IC_INT_PRIOR_3
IC_INT_PRIOR_4
IC_INT_PRIOR_5
IC_INT_PRIOR_6
IC_INT_PRIOR_7
(These bit fields are mutually exclusive)

Interrupt Sub-Priority
IC_INT_SUB_PRIOR_0
IC_INT_SUB_PRIOR_1
IC_INT_SUB_PRIOR_2
IC_INT_SUB_PRIOR_3
(These bit fields are mutually exclusive)

Return Value: None
Remarks: This function clears the Interrupt Flag bit and then sets the interrupt priority and enables/disables the interrupt.
Code Example: ConfigIntCapture1(IC_INT_ON | IC_INT_PRIOR_1 | IC_INT_SUB_PRIOR_3);
ReadCapture1
ReadCapture2
ReadCapture3
ReadCapture4
ReadCapture5

Description:  This function reads all the pending Input Capture buffers.
Include:  plib.h
Prototype:
void ReadCapture1(unsigned int *buffer);
void ReadCapture2(unsigned int *buffer);
void ReadCapture3(unsigned int *buffer);
void ReadCapture4(unsigned int *buffer);
void ReadCapture5(unsigned int *buffer);
Arguments:  buffer  This is the pointer to the locations where the data read from
the Input Capture buffers have to be stored.
Return Value:  None
Remarks:  This function reads all the pending Input Capture buffers until the
buffers are empty indicated by the ICxCON<ICBNE> bit getting
cleared.
Code Example:
unsigned int buffer[16];
ReadCapture1(buffer);
### 12.2 Individual Macros

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Include</th>
<th>Arguments</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableIntIC1</td>
<td>This macro enables the interrupt on capture event.</td>
<td>plib.h</td>
<td>None</td>
<td>This macro sets Input Capture Interrupt Enable bit of Interrupt Enable Control register.</td>
</tr>
<tr>
<td>EnableIntIC2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableIntIC3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableIntIC4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnableIntIC5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Include</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remarks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Example</td>
<td>EnableIntIC1;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Example</td>
<td>DisableIntIC1;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisableIntIC2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisableIntIC3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisableIntIC4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisableIntIC5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Include</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remarks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Example</td>
<td>DisableIntIC4;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| SetPriorityIntIC1 | This macro sets priority for input capture interrupt.                      | plib.h   | config    | Input Capture interrupt priority information as defined below:  
  * Interrupt Priority  
  | IC_INT_PRIOR_0  
  | IC_INT_PRIOR_1  
  | IC_INT_PRIOR_2  
  | IC_INT_PRIOR_3  
  | IC_INT_PRIOR_4  
  | IC_INT_PRIOR_5  
  | IC_INT_PRIOR_6  
  | IC_INT_PRIOR_7  
  *(These bit fields are mutually exclusive)* |
Interrupt Sub-Priority
- IC_INT_SUB_PRIOR_0
- IC_INT_SUB_PRIOR_1
- IC_INT_SUB_PRIOR_2
- IC_INT_SUB_PRIOR_3

(These bit fields are mutually exclusive)

Remarks: This macro sets Input Capture Interrupt Priority bits of Interrupt Priority Control register.

Code Example: SetPriorityIntIC4(IC_INT_PRIOR_5 | IC_INT_SUB_PRIOR_2);

---

**mIC1CaptureReady()**
**mIC2CaptureReady()**
**mIC3CaptureReady()**
**mIC4CaptureReady()**

Description: This macro returns true if one or more event is captured.

Include: plib.h

Arguments: None

Remarks: The mICxCaptureReady provides a status if a new capture is available.

Code Example: if( mIC1CaptureReady() )
    { // we have capture(s) ready

---

**mIC1ReadCapture()**
**mIC2ReadCapture()**
**mIC3ReadCapture()**
**mIC4ReadCapture()**

Description: This macro returns one captured timer value.

Include: plib.h

Arguments: None

Remarks: The mICxCaptureReady provides a status if a new capture is available.

Code Example: if( mIC1CaptureReady() )
    {
        // we have capture(s) ready
        capVal = mIC1ReadCapture();
    }
12.3 Example of Use

#include <plib.h>

#define FOSC 60E6
#define PB_DIV 8
#define PRESCALE 256
#define MSEC 10E-3
#define T1_TICK (500 * MSEC * FOSC)/(PB_DIV * PRESCALE)

int main(void)
{
    unsigned int CaptureTime;
    mIC1ClearIntFlag();

    // Setup Timer 3
    OpenTimer3(T3_ON | T1_PS_1_256, T1_TICK);

    // Enable Input Capture Module 1
    // - Capture Every edge
    // - Enable capture interrupts
    // - Use Timer 3 source
    // - Capture rising edge first
    OpenCapture1( IC_EVERY_EDGE | IC_INT_1CAPTURE | IC_TIMER3_SRC |
                  IC_FEDGE_RISE | IC_ON );

    // Wait for Capture events
    while( !mIC1CaptureReady() ) ;

    // Now Read the captured timer value
    while( mIC1CaptureReady() )
    {
        CaptureTime = mIC1ReadCapture();
        // process data
        // ...
    }

    CloseCapture1();
    CloseTimer3();

    while(1)
    {
    }
}
13.0 OUTPUT COMPARE FUNCTIONS

This section contains a list of individual functions for Output Compare module and an example of use of the functions. Functions may be implemented as macros.

13.1 Individual Functions

CloseOC1
CloseOC2
CloseOC3
CloseOC4
CloseOC5

Description: This function turns off the Output Compare module.
Include: plib.h
Prototype: void CloseOC1(void);
void CloseOC2(void);
void CloseOC3(void);
void CloseOC4(void);
void CloseOC5(void);
Arguments: None
Return Value: None
Remarks: This function disables the Output Compare interrupt and then turns off the module. The Interrupt Flag bit is also cleared.
Source File: Code Example: CloseOC1();
### ConfigIntOC1
### ConfigIntOC2
### ConfigIntOC3
### ConfigIntOC4
### ConfigIntOC5

| Description: | This function configures the Output Compare interrupt. |
| Include: | plib.h |
| Prototype: | void ConfigIntOC1(unsigned int config); |
| | void ConfigIntOC2(unsigned int config); |
| | void ConfigIntOC3(unsigned int config); |
| | void ConfigIntOC4(unsigned int config); |
| | void ConfigIntOC5(unsigned int config); |
| Arguments: | **config** Output Compare interrupt priority and enable/disable information as defined below: |
| | **Interrupt enable/disable** |
| | OC_INT_ON |
| | OC_INT_OFF |
| | **Interrupt Priority** |
| | OC_INT_PRIOR_0 |
| | OC_INT_PRIOR_1 |
| | OC_INT_PRIOR_2 |
| | OC_INT_PRIOR_3 |
| | OC_INT_PRIOR_4 |
| | OC_INT_PRIOR_5 |
| | OC_INT_PRIOR_6 |
| | OC_INT_PRIOR_7 |
| | **Interrupt Sub-priority** |
| | OC_INT_SUB_PRIOR_0 |
| | OC_INT_SUB_PRIOR_1 |
| | OC_INT_SUB_PRIOR_2 |
| | OC_INT_SUB_PRIOR_3 |
| Return Value: | None |
| Remarks: | This function clears the Interrupt Flag bit and then sets the interrupt priority and enables/disables the interrupt. |
| Source File: | |
| Code Example: | ConfigIntOC1(OC_INT_ON | OC_INT_PRIOR_2 | OC_INT_SUB_PRIOR_2); |
OpenOC1
OpenOC2
OpenOC3
OpenOC4
OpenOC5

Description: This function configures the Output Compare module.

Include: plib.h

Prototype:
void OpenOC1(unsigned int config, unsigned int value1, unsigned int value2);
void OpenOC2(unsigned int config, unsigned int value1, unsigned int value2);
void OpenOC3(unsigned int config, unsigned int value1, unsigned int value2);
void OpenOC4(unsigned int config, unsigned int value1, unsigned int value2);
void OpenOC5(unsigned int config, unsigned int value1, unsigned int value2);

Arguments:
config This contains the parameters to be configured in the OCxCON register as defined below:
Module on/off control
OC_ON
OC_OFF

Idle mode operation
OC_IDLE_STOP
OC_IDLE_CON

Timer width select
OC_TIMER_Mode32
OC_TIMER_Mode16

Clock select
OC_TIMER2_SRC
OC_TIMER3_SRC

Output Compare modes of operation
OC_PWM_FAULT_PIN_ENABLE
OC_PWM_FAULT_PIN_DISABLE
OC_CONTINUE_PULSE
OC_SINGLE_PULSE
OC_TOGGLE_PULSE
OC_HIGH_LOW
OC_LOW_HIGH
OC_MODE_OFF

value1 This contains the value to be stored into OCxRS Secondary Register.

value2 This contains the value to be stored into OCxR Main Register.

Return Value: None
Remarks: This function configures the Output Compare Module Control register (OCxCON) with the following parameters: Clock select, mode of operation, operation in Idle mode. It also configures the OCxRS and OCxR registers.

Code Example:
```c
OpenOC1(OC_ON | OC_TIMER2_SRC |
OC_FWM_FAULT_PIN_ENABLE, 0x80, 0x60);
```
### ReadDCOC1PWM
### ReadDCOC2PWM
### ReadDCOC3PWM
### ReadDCOC4PWM
### ReadDCOC5PWM

<table>
<thead>
<tr>
<th>Description:</th>
<th>This function reads the duty cycle from the Output Compare Secondary register.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>plib.h</td>
</tr>
</tbody>
</table>
| Prototype:  | `unsigned int ReadDCOC1PWM(void);`  
`unsigned int ReadDCOC2PWM(void);`  
`unsigned int ReadDCOC3PWM(void);`  
`unsigned int ReadDCOC4PWM(void);`  
`unsigned int ReadDCOC5PWM(void);` |
| Arguments:  | None                                                                               |
| Return Value: | This function returns the content of OCxRS register when Output Compare module is in PWM mode. Else ‘-1’ is returned |
| Remarks:    | This function reads the duty cycle from the Output Compare Secondary register (OCxRS) when Output Compare module is in PWM mode. If not in PWM mode, the functions returns a value of ‘-1’. |

#### Code Example:
```c
unsigned int compare_reg;
compare_reg = ReadDCOC1PWM();
```
ReadRegOC1
ReadRegOC2
ReadRegOC3
ReadRegOC4
ReadRegOC5

Description: This function reads the duty cycle registers when Output Compare module is not in PWM mode.

Include: plib.h

Prototype:
unsigned int ReadRegOC1(unsigned int reg);
unsigned int ReadRegOC2(unsigned int reg);
unsigned int ReadRegOC3(unsigned int reg);
unsigned int ReadRegOC4(unsigned int reg);
unsigned int ReadRegOC5(unsigned int reg);

Arguments: 
reg This indicates if the read should happen from the main or secondary duty cycle registers of Output Compare module.
  If reg is '1', then the contents of Main Duty Cycle register (OCxR) is read.
  If reg is '0', then the contents of Secondary Duty Cycle register (OCxRS) is read.

Return Value: If reg is '1', then the contents of Main Duty Cycle register (OCxR) is read.
  If reg is '0', then the contents of Secondary Duty Cycle register (OCxRS) is read.
  If Output Compare module is in PWM mode, '-1' is returned.

Remarks: The read of Duty Cycle register happens only when Output Compare module is not in PWM mode. Else, a value of '-1' is returned.

Code Example: unsigned int dutycycle_reg;
dutycycle_reg = ReadRegOC1(1);
SetDCOC1PWM
SetDCOC2PWM
SetDCOC3PWM
SetDCOC4PWM
SetDCOC5PWM

Description: This function configures the Output Compare Secondary Duty Cycle register (OCxRS) when the module is in PWM mode.

Include: outcompare.h

Prototype:

void SetDCOC1PWM(unsigned int dutycycle);
void SetDCOC2PWM(unsigned int dutycycle);
void SetDCOC3PWM(unsigned int dutycycle);
void SetDCOC4PWM(unsigned int dutycycle);
void SetDCOC5PWM(unsigned int dutycycle);

Arguments: dutycycle This is the duty cycle value to be stored into Output Compare Secondary Duty Cycle register (OCxRS).

Return Value: None

Remarks: The Output Compare Secondary Duty Cycle register (OCxRS) will be configured with new value only if the module is in PWM mode.

Code Example: SetDCOC1PWM(dutycycle);
SetPulseOC1
SetPulseOC2
SetPulseOC3
SetPulseOC4
SetPulseOC5

Description: This function configures the Output Compare main and secondary registers (OCxR and OCxRS) when the module is not in PWM mode.

Include: plib.h

Prototype:

```c
void SetPulseOC1(unsigned int pulse_start, unsigned int pulse_stop);
void SetPulseOC2(unsigned int pulse_start, unsigned int pulse_stop);
void SetPulseOC3(unsigned int pulse_start, unsigned int pulse_stop);
void SetPulseOC4(unsigned int pulse_start, unsigned int pulse_stop);
void SetPulseOC5(unsigned int pulse_start, unsigned int pulse_stop);
```

Arguments:

- **pulse_start**  This is the value to be stored into Output Compare Main register (OCxR).
- **pulse_stop**  This is the value to be stored into Output Compare Secondary register (OCxRS).

Return Value: None

Remarks: The Output Compare duty cycle registers (OCxR and OCxRS) will be configured with new values only if the module is not in PWM mode.

Code Example:

```c
pulse_start = 0x40;
pulse_stop  = 0x60;
SetPulseOC1(pulse_start, pulse_stop);
```
#include <plib.h>

/* This is ISR corresponding to OC1 interrupt */
#pragma interrupt OC1Interrupt ipl2 vector 6
void OC1Interrupt(void)
{
    IFS0bits.OC1IF = 0;
}

int main(void)
{
    /* Holds the value at which OCx Pin to be driven high */
    unsigned int pulse_start;
    /* Holds the value at which OCx Pin to be driven low */
    unsigned int pulse_stop;
    /* Turn off OC1 module */
    CloseOC1;
    /* Configure output compare1 interrupt */
    ConfigIntOC1(OC_INT_PRIOR_5 | EXT_INT_SUB_PRI_2);
    /* Configure OC1 module for required pulse width */
    pulse_start = 0x40;
    pulse_stop = 0x60;
    PR2 = 0x80 ;
    T2CON = 0x8000;
    /* Configure Output Compare module to 'initialise OCx pin
    low and generate continuous pulse'mode */
    OpenOC1(OC_IDLE_CON | OC_TIMER2_SRC | 
     OC_CONTINUE_PULSE,
     pulse_stop, pulse_start);
    /* Generate continuous pulse till TMR2 reaches 0xff00 */
    while(TMR2<= 0xff00);
    asm("nop");
    CloseOC1;
    return 0;
}
14.0 SPI FUNCTIONS

This section provides a list and a description of the interface functions that are part of the SPI API Peripheral Library.

14.1 Open Functions

These functions deal with the initialization of the SPI channel.

**OpenSPI1**

**OpenSPI2**

| Description: | These functions initialize and enable the SPI modules. |
| Include:     | plib.h |
| Prototype:   | void OpenSPI1(unsigned int config1, unsigned int config2);  
|             | void OpenSPI2(unsigned int config1, unsigned int config2);  |
| Arguments:   | config1 This contains the parameters to be configured in the SPIxCON register as defined below:  
|             | **Framed SPI support Enable/Disable**  
|             | FRAME_ENABLE_ON  
|             | FRAME_ENABLE_OFF |
|             | **Frame Sync Pulse direction control**  
|             | FRAME_SYNC_INPUT  
|             | FRAME_SYNC_OUTPUT |
|             | **SDO Pin Control bit**  
|             | DISABLE_SDO_PIN  
|             | ENABLE_SDO_PIN |
|             | **Word/Byte Communication mode**  
|             | SPI_MODE32_ON  
|             | SPI_MODE32_OFF  
|             | SPI_MODE16_ON  
|             | SPI_MODE16_OFF |
|             | **SPI Data Input Sample phase**  
|             | SPI_SMP_ON  
|             | SPI_SMP_OFF  
|             | **SPI Clock Edge Select**  
|             | SPI_CKE_ON  
|             | SPI_CKE_OFF |
|             | **SPI slave select enable**  
|             | SLAVE_ENABLE_ON  
|             | SLAVE_ENABLE_OFF |
|             | **SPI Clock polarity select**  
|             | CLK_POL_ACTIVE_LOW  
|             | CLK_POL_ACTIVE_HIGH |
|             | **SPI Mode Select bit**  
|             | MASTER_ENABLE_ON  
|             | MASTER_ENABLE_OFF |
# OpenSPI1 (Continued)

## OpenSPI2

### Secondary Prescale select
- SEC_PRESCAL_1_1
- SEC_PRESCAL_2_1
- SEC_PRESCAL_3_1
- SEC_PRESCAL_4_1
- SEC_PRESCAL_5_1
- SEC_PRESCAL_6_1
- SEC_PRESCAL_7_1
- SEC_PRESCAL_8_1

### Primary Prescale select
- PRI_PRESCAL_1_1
- PRI_PRESCAL_4_1
- PRI_PRESCAL_16_1
- PRI_PRESCAL_64_1

### config

This contains the parameters to be configured in the SPIxCON and SPIxSTAT registers as defined below:

- **SPI Enable/Disable**
  - SPI_ENABLE
  - SPI_DISABLE

- **SPI Operation in Debug Mode**
  - SPI_FRZ_BREAK
  - SPI_FRZ_CONTINUE

- **SPI Idle mode operation**
  - SPI_IDLE_CON
  - SPI_IDLE_STOP

- **Receive Overflow Flag bit**
  - SPI_RX_OVFLOW
  - SPI_RX_OVFLOW_CLR

- **Frame pulse polarity selection**
  - FRAME_POL_ACTIVE_HIGH
  - FRAME_POL_ACTIVE_LOW

- **Frame pulse coincidence selection**
  - FRAME_SYNC_EDGE_COINCIDE
  - FRAME_SYNC_EDGE_PRECEDE

### Return Value:
None

### Remarks:
1. SpiOpenConfig1::PPRE and SpiOpenConfig1::SPRE fields are used only for backward compatibility reasons only. They don't correspond to physical bits into the SPI control register.
2. When selecting the number of bits per character, MODE32 has the highest priority. If MODE32 is not set, then MODE16 selects the character width.
3. The format of configuration words is chosen for backward compatibility reasons. The config words don't reflect the actual register bits.

### Source File:
- spi_open_spil_lib.c
- spi_open_spi2_lib.c

### Code Example:
OpenSPI1(SPI_MODE32_ON|SPI_SMP_ON|MASTER_ENABLE_ON|SEC_PRESCAL_1_1|PRI_PRESCAL_1_1, SPI_ENABLE);

## 14.2 Close Functions

These functions close an opened SPI channel.
14.3 Interrupt configuration Functions

These functions configure the interrupts for a SPI channel.

### CloseSPI1
### CloseSPI2

**Description:** This routines disable the SPI modules and clear the interrupt bits.

**Include:** plib.h

**Prototype:**

```c
void CloseSPI1(void);
void CloseSPI2(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** None

**Source File:** plib.h

**Code Example:**

```c
CloseSPI1();
```

### ConfigIntSPI1
### ConfigIntSPI2

**Description:** These functions configure Interrupt and set the Interrupt Priority.

**Include:** plib.h

**Prototype:**

```c
void ConfigIntSPI1( unsigned int config);
void ConfigIntSPI2( unsigned int config);
```

**Arguments:**

- `config` This contains the interrupt parameters to be configured as defined below:
  - **SPI Fault Interrupt Enable/Disable**
    - `SPI_FAULT_INT_EN`
    - `SPI_FAULT_INT_DIS`
  - **SPI Transmit Interrupt Enable/Disable**
    - `SPI_TX_INT_EN`
    - `SPI_TX_INT_DIS`
  - **SPI Receive Interrupt Enable/Disable**
    - `SPI_RX_INT_EN`
    - `SPI_RX_INT_DIS`
  - **SPI Interrupt Sub-priority**
    - `SPI_INT_SUB_PRI_0`
    - `SPI_INT_SUB_PRI_1`
    - `SPI_INT_SUB_PRI_2`
    - `SPI_INT_SUB_PRI_3`
### ConfigIntSPI1

**Description:**
These macros configure the interrupt priority of SPI 1.

**Prototype:**

```c
void ConfigIntSPI1(uint32_t enable);```

**Arguments:**

- `enable`: A combination of SPI interrupt enable flags.

**Return Value:**
None

**Remarks:**
None

**Source File:**
plib.h

**Code Example:**

```c
ConfigIntSPI1(SPI_RX_INT_EN|SPI_TX_INT_EN|SPI_INT_PRI_3);
```

### ConfigIntSPI2

**Description:**
These macros configure the interrupt priority of SPI 2.

**Prototype:**

```c
void ConfigIntSPI2(uint32_t enable);```

**Arguments:**

- `enable`: A combination of SPI interrupt enable flags.

**Return Value:**
None

**Remarks:**
None

**Source File:**
plib.h

**Code Example:**

```c
ConfigIntSPI2(SPI_RX_INT_EN|SPI_TX_INT_EN|SPI_INT_PRI_3);
```

### SPI Interrupt Priority

The interrupt priorities are defined as follows:

- SPI_INT_PRI_0
- SPI_INT_PRI_1
- SPI_INT_PRI_2
- SPI_INT_PRI_3
- SPI_INT_PRI_4
- SPI_INT_PRI_5
- SPI_INT_PRI_6
- SPI_INT_PRI_7

### EnableIntSPI1

**Description:**
These macros enable the receive and transmit interrupts for SPI 1.

**Prototype:**

```c
void EnableIntSPI1(void);```

**Arguments:**
None

**Return Value:**
None

**Remarks:**
None

**Source File:**
plib.h

**Code Example:**

```c
EnableIntSPI1;
```

### EnableIntSPI2

**Description:**
These macros enable the receive and transmit interrupts for SPI 2.

**Prototype:**

```c
void EnableIntSPI2(void);```

**Arguments:**
None

**Return Value:**
None

**Remarks:**
None

**Source File:**
plib.h

**Code Example:**

```c
EnableIntSPI2;
```

### DisableIntSPI1

**Description:**
These macros disable the receive and transmit interrupts for SPI 1.

**Prototype:**

```c
void DisableIntSPI1(void);```

**Arguments:**
None

**Return Value:**
None

**Remarks:**
None

**Source File:**
plib.h

**Code Example:**

```c
DisableIntSPI1;
```

### DisableIntSPI2

**Description:**
These macros disable the receive and transmit interrupts for SPI 2.

**Prototype:**

```c
void DisableIntSPI2(void);```

**Arguments:**
None

**Return Value:**
None

**Remarks:**
None

**Source File:**
plib.h

**Code Example:**

```c
DisableIntSPI2;
```
### DisableIntSPI1
### DisableIntSPI2

**Code Example:**
```
DisableIntSPI1;
DisableIntSPI2;
```

### SetPriorityIntSPI1
### SetPriorityIntSPI2

**Description:** These functions set the interrupt priority for SPI channel 1, 2.

**Include:** `plib.h`

**Prototype:**
```
void SetPriorityIntSPI1(int priority);
void SetPriorityIntSPI2(int priority);
```

**Arguments:**
- `priority`: interrupt priority for the SPI channel:
  - SPI Interrupt Priority
    - `SPI_INT_PRI_0`
    - `SPI_INT_PRI_1`
    - `SPI_INT_PRI_2`
    - `SPI_INT_PRI_3`
    - `SPI_INT_PRI_4`
    - `SPI_INT_PRI_5`
    - `SPI_INT_PRI_6`
    - `SPI_INT_PRI_7`

**Return Value:** None

**Remarks:** None

**Source File:** `plib.h`

**Code Example:**
```
SetPriorityIntSPI1(SPI_INT_PRI_0);
SetPriorityIntSPI2(SPI_INT_PRI_3);
```

### SetSubPriorityIntSPI1
### SetSubPriorityIntSPI2

**Description:** These functions set the interrupt sub-priority for SPI channel 1, 2.

**Include:** `plib.h`

**Prototype:**
```
void SetSubPriorityIntSPI1(int subPriority);
void SetSubPriorityIntSPI2(int subPriority);
```

**Arguments:**
- `subPriority`: interrupt sub-priority for the SPI channel:
  - SPI Interrupt Sub-priority
    - `SPI_INT_SUB_PRI_0`
    - `SPI_INT_SUB_PRI_1`
    - `SPI_INT_SUB_PRI_2`
    - `SPI_INT_SUB_PRI_3`

**Return Value:** None

**Remarks:** None

**Source File:** `plib.h`

**Code Example:**
```
SetSubPriorityIntSPI1(SPI_INT_SUB_PRI_3);
SetSubPriorityIntSPI2(SPI_INT_SUB_PRI_1);
```
14.4  Read Write access Functions

These functions read or write data from/to a SPI channel.

**DataRdySPI1**
**DataRdySPI2**

| Description: | These functions determine if there is a data to be read from the SPIBUF register. |
| Include:     | plib.h |
| Prototype:   | int DataRdySPI1(void); int DataRdySPI2(void); |
| Arguments:   | None |
| Return Value:| true if data is available (Receiver Buffer Full), false otherwise |
| Remarks:     | None |
| Source File: | plib.h |
| Code Example:| int isDataAvlbl; isDataAvlbl = DataRdySPI1(); |

**TxBufFullSPI1**
**TxBufFullSPI2**

| Description: | These functions test if transmit buffer is full and determine if the data can be written to the SPIBUF register without overwriting the previous, unsent data. |
| Include:     | plib.h |
| Prototype:   | int TxBufFullSPI1(void); int TxBufFullSPI2(void); |
| Arguments:   | None |
| Return Value:| - true if SPI buffer is full and data cannot be written to device, in order to be serialized - false otherwise |
| Remarks:     | None |
| Source File: | plib.h |
| Code Example:| if(!TxBufFullSPI1())WriteSPI1('a'); |

**ReadSPI1**
**ReadSPI2**

| Description: | This function will read single byte/half word/word from SPI receive register. |
| Include:     | plib.h |
| Prototype:   | unsigned int ReadSPI1(void); unsigned int ReadSPI2(void); |
| Arguments:   | None |
ReadSPI1 (Continued)
ReadSPI2

Return Value: Returns the contents of SPIBUF register in byte/hword/word format.
Remarks: None
Source File: plib.h
Code Example: int data=ReadSPI1();

WriteSPI1
WriteSPI2

Description: This function writes the data to be transmitted into the Transmit Buffer (SPIxBUF) register.
Include: plib.h
Prototype: void WriteSPI1(unsigned int data);
void WriteSPI2(unsigned int data);
Arguments: data This is the data to be transmitted which will be stored in SPI buffer.
Remarks: This function writes the data (byte/half word/word) to be transmitted into the transmit buffer, depending on the current communication mode: 8, 16 or 32 bits.
Return Value: None
Source File: plib.h
Code Example: WriteSPI1(0x44332211);

getcSPI1
getcSPI2

Description: This function waits for receive data to be available. It will then read single byte/half word/word from the SPI channel.
Include: plib.h
Prototype: unsigned int getcSPI1(void);
unsigned int getcSPI2(void);
Arguments: None
Remarks: The byte/half word/word accesses will perform correctly, depending on the current communication mode: 8, 16 or 32 bits.
Return Value: None
Source File: spi_getc_spi1_lib.c
spi_getc_spi2_lib.c
Code Example: int data=getcSPI1();
### putcSPI1, putcSPI2

**Description:** This routine writes a single byte/half word/word to the SPI bus. It waits so that it doesn’t overwrite the previous untransmitted data.

**Include:**

```c
plib.h
```

**Prototype:**

```c
void putcSPI1(unsigned int data_out);
void putcSPI2(unsigned int data_out);
```

**Arguments:**

- `data_out`: This is the data to be transmitted over the SPI channel.

**Remarks:** The byte/half word/word accesses will perform correctly, depending on the current communication mode: 8, 16 or 32 bits.

**Return Value:** None

**Source File:**

```c
plib.h
```

**Code Example:**

```c
putcSPI1(0xaa);
```

### getsSPI1, getsSPI2

**Description:** This routine reads a string from the SPI receive buffer. The number of characters (bytes/half words/words) to be read is determined by parameter ‘length’.

**Include:**

```c
plib.h
```

**Prototype:**

```c
unsigned int getsSPI1(unsigned int length, unsigned int *rdptr, unsigned int spi_data_wait);
unsigned int getsSPI2(unsigned int length, unsigned int *rdptr, unsigned int spi_data_wait);
```

**Arguments:**

- `length`: This is the number of characters to be received.
- `rdptr`: This is the pointer to the location where the data received have to be stored.
- `spi_data_wait`: This is a retries count for which the function has to poll the SPI channel for having data ready before quitting.

**Remarks:** `rdptr` is considered to be 8/16/32 bits data pointer, according to the current SPI mode.

**Return Value:** Number of data bytes yet to be received

**Source File:**

```c
spi_gets_spi1_lib.c
spi_gets_spi2_lib.c
```

**Code Example:**

```c
unsigned char buff[100];
getsSPI1(sizeof(buff), buff, 1000);
```
14.5  Channel parameterized Functions
These functions have the required SPI channel as a function parameter.

14.5.1  OPEN/CLOSE AND CONFIGURATION FUNCTIONS

SpiChnOpen

Description: This function initializes the SPI channel and also sets the brg register.

Include: plib.h

Prototype: void SpiChnOpen(int chn, unsigned int config, UINT fpbDiv)

Arguments: chn This is the number of the SPI channel: 1 or 2
config This contains the configuration parameters for the SPIxCON register as defined below:

- Master mode Enable
  SPICON_MSTEN
- Clock Polarity control
  SPICON_CKP
- Slave Select pin control
  SPICON_SSEN
- Clock Edge control
  SPICON_CKE

Source File: spi_spi1_lib.c
spi_spi2_lib.c

Code Example: char* myBuff="This is data transmitted over SPI";
putsSPI1(strlen(myBuff), myBuff);
**SpiChnOpen**

- **Sample phase control**
  SPICON_SMP

- **Character width control**
  SPICON_MODE16
  SPICON_MODE32

- **SDO pin control**
  SPICON_DISSDO

- **Idle functionality control**
  SPICON_SIDL

- **Debug functionality control**
  SPICON_FRZ

- **Module ON control**
  SPICON_ON

- **Frame Sync edge control**
  SPICON_SPIFE

- **Frame Sync Polarity control**
  SPICON_FRMPOL

- **Frame Sync Direction control**
  SPICON_FRMSYNC

- **Frame Mode enable**
  SPICON_FRMEN

**fpbDiv**  This is the Fpb divisor to extract the baud rate: \( BR = \frac{Fpb}{fpbDiv} \).

**Remarks:**
- The SPI baudrate \( BR \) is given by:
  \[ BR = \frac{Fpb}{2 \times (SPIBRG+1)} \]

  - The input parameters \( fpbDiv \) specifies the Fpb divisor term \( 2 \times (SPIBRG+1) \), so the BRG is calculated as \( SPIBRG = \frac{fpbDiv}{2} - 1 \).
  - The baud rate is always obtained by dividing the Fpb to an even number between 2 and 1024.

- When selecting the character width, SPICON_MODE32 has the highest priority. If SPICON_MODE32 is not set, then SPICON_MODE16 selects the character width.

**Return Value:** None

**Source File:** spi_chn_open_lib.c

**Code Example:**
```
SpiChnOpen(1,
          SPICON_MSTEN|SPICON_SMP|SPICON_MODE32|SPICON_ON, 4);
```

---

**SpiChnClose**

**Description:** This function closes the SPI channel. Some previous error conditions are cleared. Channel interrupts are disabled.

**Include:** plib.h

**Prototype:**
```
void SpiChnClose(int chn);
```

**Arguments:**
- **chn**  This is the number of the SPI channel: 1 or 2

**Remarks:** None

**Return Value:** None

**Source File:** spi_chn_close_lib.c

**Code Example:**
```
SpiChnClose(2);
```
SpiChnSetBrg, mSpiChnSetBrg

Description: This function/macro updates the values for the SPI channel baud rate generator register

Include: plib.h

Prototype: void SpiChnSetBrg(int chn, UINT brg);

Arguments:
- chn This is the number of the SPI channel: 1 or 2
- brg value for the brg register

Remarks: The SPI baudrate BR is given by:
BR=Fpb/(2*(SPIBRG+1))
The baud rate is always obtained by dividing the Fpb to an even number between 2 and 1024.

Return Value: None

Source File: spi_chn_set_brg_lib.c

Code Example:
int chn=1; SpiChnSetBrg(chn, 4);
or
mSpiChnSetBrg(1, 4);

SpiChnChgMode

Description: This function changes the SPI channel mode on the fly.

Include: plib.h

Prototype: void SpiChnChgMode(int chn, int isMaster, int isFrmMaster);

Arguments:
- chn This is the number of the SPI channel: 1 or 2
- isMaster switching to master mode required
- isFrmMaster switching to frame master mode required

Remarks: When changing mode, the function blocks until the current transfer, if any, is completed.

Remarks: None

Remarks: isFrmMaster is relevant only if the SPI channel is operating in frame mode.

Return Value: None

Source File: spi_chn_chg_mode_lib.c

Code Example: SpiChnChgMode(1, 1, 1);

14.5.2 DATA TRANSFER FUNCTIONS
### SpiChnDataRdy

**Description:**
This function reads the SPI channel data ready condition.

**Include:**
plib.h

**Prototype:**
```c
int SpiChnDataRdy(int chn);
```

**Arguments:**
- **chn**
  This is the number of the SPI channel: 1 or 2

**Remarks:**
None

**Return Value:**
- TRUE- if data available
- FALSE otherwise

**Source File:**
spi_chn_data_rdy_lib.c

**Code Example:**
```c
int isDataAvlbl=SpiChnDataRdy(1);
```

### SpiChnGetC

**Description:**
This function waits for data to be available and returns it.

**Include:**
plib.h

**Prototype:**
```c
int SpiChnGetC(int chn);
```

**Arguments:**
- **chn**
  This is the number of the SPI channel: 1 or 2

**Remarks:**
None

**Return Value:**
data available in the SPI rx buffer

**Source File:**
spi_chn_getc_lib.c

**Code Example:**
```c
int newData=SpiChnGetC(2);
```

### SpiChnGetS

**Description:**
This routine reads a buffer of characters from the corresponding SPI channel receive buffer.

**Include:**
plib.h

**Prototype:**
```c
void SpiChnGetS(int chn, unsigned int *pBuff, unsigned int nChars);
```

**Arguments:**
- **chn**
  This is the number of the SPI channel: 1 or 2
- **pBuff**
  address of buffer to store data
- **nChars**
  number of byte/half word/word characters expected

**Remarks:**
pBuff has to be a valid pointer to a buffer large enough to store all the received characters
pBuff is considered to be 8/16/32 bits data pointer, according to the current SPI mode
The function blocks waiting for the whole buffer to be received.

**Return Value:**
data available in the SPI rx buffer

**Source File:**
spi_chn_gets_lib.c

**Code Example:**
```c
unsigned short myBuff[100];
SpiChnGetS(2, myBuff, sizeof(myBuff)/sizeof(*myBuff));  // receive 16 bit characters
```
### SpiChnTxBuffEmpty

**Description:** This function reads the SPI channel transmit buffer empty condition.

**Include:** plib.h

**Prototype:**

```c
int SpiChnTxBuffEmpty(int chn);
```

**Arguments:**

- `chn` This is the number of the SPI channel: 1 or 2

**Remarks:** None

**Return Value:**

- TRUE if transmit buffer empty
- FALSE otherwise

**Source File:** spi_chn_tx_buff_empty_lib.c

**Code Example:**

```c
int canTransmit=SpiChnTxBuffEmpty(1);
```

### SpiChnPutC

**Description:** This routine writes a single byte/half word/word to the SPI channel. It waits for TX buffer empty, so that it doesn't overwrite the previous untransmitted data.

**Include:** plib.h

**Prototype:**

```c
void SpiChnPutC(int chn, int data);
```

**Arguments:**

- `chn` This is the number of the SPI channel: 1 or 2
- `data` the data to be written to the SPI channel

**Remarks:** Byte/half word/word accesses will perform correctly based on the current SPI channel configuration.

**Return Value:** None

**Source File:** spi_chn_putc_lib.c

**Code Example:**

```c
SpiChnPutC(1, 0x1b);  // send an ESC character
```

### SpiChnPutS

**Description:** This function writes the specified number of 8/16/32 bit characters from the specified buffer. It waits for Tx buffer empty so the characters are not overwritten.

**Include:** plib.h

**Prototype:**

```c
void SpiChnPutS(int chn, unsigned int* pBuff, unsigned int nChars);
```

**Arguments:**

- `chn` This is the number of the SPI channel: 1 or 2
- `pBuff` address of buffer storing the data to be transmitted.
- `nChars` number of byte/half word/word characters to be transmitted.

**Remarks:**

- `pBuff` is considered to be 8/16/32 bits data pointer, according to the current SPI mode.

**Return Value:** None

**Source File:** spi_chn_puts_lib.c

**Code Example:**

```c
SpiChnPutS(1, myBuff, 100);
```
### SpiChnGetRov

**Description:** This function reads the SPI channel overflow condition and clears it, if required

**Include:** plib.h

**Prototype:**

```c
int SpiChnGetRov(int chn, int clear);
```

**Arguments:**

- `chn` This is the number of the SPI channel: 1 or 2
- `clear` if TRUE, the overflow condition has to be cleared, if present

**Remarks:** None

**Return Value:** None

**Source File:** spi_chn_get_rov_lib.c

**Code Example:**

```c
int isOvfl=SpiChnGetRov(1, FALSE);
```

### 14.5.3 INTERRUPT FLAGS FUNCTIONS

#### SpiChnGetRovIntFlag

**mSpiChnGetRovIntFlag**

**Description:** This function/macro reads the SPI channel overflow interrupt flag.

**Include:** plib.h

**Prototype:**

```c
int SpiChnGetRovIntFlag(int chn);
```

**Arguments:**

- `chn` This is the number of the SPI channel: 1 or 2

**Remarks:** None

**Return Value:** None

**Source File:** plib.h

**Code Example:**

```c
int chn=2; int isOvflFlag=SpiChnGetRovIntFlag(chn);
int isOvflFlag=mSpiChnGetRovIntFlag(1);
```

#### SpiChnClrRovIntFlag

**mSpiChnClrRovIntFlag**

**Description:** This function/macro clears the SPI channel overflow interrupt flag.

**Include:** plib.h

**Prototype:**

```c
void SpiChnClrRovIntFlag(int chn);
```

**Arguments:**

- `chn` This is the number of the SPI channel: 1 or 2

**Remarks:** None

**Return Value:** None

**Source File:** plib.h

**Code Example:**

```c
int chn=2; SpiChnClrRovIntFlag(chn);
mSpiChnClrRovIntFlag(2);
```
### SpiChnGetRxIntFlag

**Description:**
This function/macro reads the SPI channel receive interrupt flag.

**Include:**
plib.h

**Prototype:**
```c
void SpiChnGetRxIntFlag(int chn);
```

**Arguments:**
chn  This is the number of the SPI channel: 1 or 2

**Remarks:**
None

**Return Value:**
- TRUE- if SPI Rx flag
- FALSE- otherwise

**Source File:**
plib.h

**Code Example:**
```c
int chn=1; int isRxEvent=SpiChnGetRxIntFlag(chn);
isRxEvent=mSpiChnGetRxIntFlag(1);
```

---

### SpiChnClrRxIntFlag

**Description:**
This function/macro clears the SPI channel receive interrupt flag.

**Include:**
plib.h

**Prototype:**
```c
void SpiChnClrRxIntFlag(int chn);
```

**Arguments:**
chn  This is the number of the SPI channel: 1 or 2

**Remarks:**
None

**Return Value:**
None

**Source File:**
plib.h

**Code Example:**
```c
int chn=1; SpiChnClrRxIntFlag(chn);
mSpiChnClrRxIntFlag(1);
```

---

### SpiChnGetTxIntFlag

**Description:**
This function/macro reads the SPI channel transmit interrupt flag.

**Include:**
plib.h

**Prototype:**
```c
void SpiChnGetTxIntFlag(int chn);
```

**Arguments:**
chn  This is the number of the SPI channel: 1 or 2

**Remarks:**
None

**Return Value:**
- TRUE- if SPI Tx flag
- FALSE- otherwise

**Source File:**
plib.h

**Code Example:**
```c
int chn=1; int isTxEvent=SpiChnGetTxIntFlag(chn);
isTxEvent=mSpiChnGetTxIntFlag(1);
```
### SpiChnClrTxIntFlag

**mSpiChnClrTxIntFlag**

**Description:** This function/macro clears the SPI channel transmit interrupt flag.

**Include:** plib.h

**Prototype:**

```c
void SpiChnClrTxIntFlag(int chn);
```

**Arguments:**

- *chn* This is the number of the SPI channel: 1 or 2

**Remarks:** None

**Return Value:** None

**Source File:** plib.h

**Code Example:**

```c
int chn=1; SpiChnClrTxIntFlag(chn);
```

```c
mSpiChnClrTxIntFlag(1);
```

### SpiChnGetIntFlag

**mSpiChnGetIntFlag**

**Description:** This function/macro reads the SPI channel transmit/receive or overflow interrupt flag.

**Include:** plib.h

**Prototype:**

```c
void SpiChnGetIntFlag(int chn);
```

**Arguments:**

- *chn* This is the number of the SPI channel: 1 or 2

**Remarks:** None

**Return Value:**

- **TRUE** - if SPI Tx/Rx/Ovfl flag set
- **FALSE** - otherwise

**Source File:** plib.h

**Code Example:**

```c
int chn=1; int isSpiEvent=SpiChnGetIntFlag(chn);
```

```c
isSpiEvent=mSpiChnGetIntFlag(1);
```

### SpiChnClrIntFlags

**mSpiChnClrIntFlags**

**Description:** This function/macro clears all the SPI channel interrupt flags (Tx, Rx or ovfl).

**Include:** plib.h

**Prototype:**

```c
void SpiChnClrlntFlags(int chn);
```

**Arguments:**

- *chn* This is the number of the SPI channel: 1 or 2

**Remarks:** None

**Return Value:** None

**Source File:** plib.h

**Code Example:**

```c
int chn=1; SpiChnClrlntFlags(chn);
```

```c
mSpiChnClrlntFlags(1);
```
14.5.4 INTERRUPT ENABLE/DISABLE FUNCTIONS

**SpiChnRxIntEnable**

**mSpiChnRxIntEnable**

**Description:** This function/macro enables the SPI channel receive interrupts.

**Include:** plib.h

**Prototype:**

```c
void SpiChnRxIntEnable(int chn);
```

**Arguments:**

```c
chn This is the number of the SPI channel: 1 or 2
```

**Remarks:** Clears existing interrupt flags.

**Return Value:** None

**Source File:** plib.h

**Code Example:**

```c
int chn=1; SpiChnSetRxIntEnable(chn);
mSpiChnRxIntEnable(1);
```

**SpiChnRxIntDisable**

**mSpiChnRxIntDisable**

**Description:** This function/macro disables the SPI channel receive interrupts.

**Include:** plib.h

**Prototype:**

```c
void SpiChnRxIntDisable(int chn);
```

**Arguments:**

```c
chn This is the number of the SPI channel: 1 or 2
```

**Remarks:** Clears existing interrupt flags.

**Return Value:** None

**Source File:** plib.h

**Code Example:**

```c
int chn=1; SpiChnRxIntDisable(chn);
mSpiChnRxIntDisable(1);
```

**SpiChnTxIntEnable**

**mSpiChnTxIntEnable**

**Description:** This function/macro enables the SPI channel transmit interrupts.

**Include:** plib.h

**Prototype:**

```c
void SpiChnTxIntEnable(int chn);
```

**Arguments:**

```c
chn This is the number of the SPI channel: 1 or 2
```

**Remarks:** Clears existing interrupt flags.

**Return Value:** None

**Source File:** plib.h

**Code Example:**

```c
int chn=1; SpiChnTxIntEnable(chn);
mSpiChnTxIntEnable(1);
```
### SpiChnTxIntDisable

**mSpiChnTxIntDisable**

**Description:** This function/macro disables the SPI channel transmit interrupts.

**Include:** `plib.h`

**Prototype:** `void SpiChnTxIntDisable(int chn);`

**Arguments:** `chn` This is the number of the SPI channel: 1 or 2

**Remarks:** Clears existing interrupt flags.

**Return Value:** None

**Source File:** `plib.h`

**Code Example:**

```c
int chn=1; SpiChnTxIntDisable(chn);
mSpiChnTxIntDisable(chn);
```

### SpiChnRxTxIntEnable

**mSpiChnRxTxIntEnable**

**Description:** This function/macro enables the SPI channel transmit and receive interrupts.

**Include:** `plib.h`

**Prototype:** `void SpiChnRxTxIntEnable(int chn);`

**Arguments:** `chn` This is the number of the SPI channel: 1 or 2

**Remarks:** Clears existing interrupt flags.

**Return Value:** None

**Source File:** `plib.h`

**Code Example:**

```c
int chn=1; SpiChnRxTxIntEnable(chn);
mSpiChnRxTxIntEnable(chn);
```

### SpiChnRxTxIntDisable

**mSpiChnRxTxIntDisable**

**Description:** This function/macro disables the SPI channel transmit and receive interrupts.

**Include:** `plib.h`

**Prototype:** `void SpiChnRxTxIntDisable(int chn);`

**Arguments:** `chn` This is the number of the SPI channel: 1 or 2

**Remarks:** Clears existing interrupt flags.

**Return Value:** None

**Source File:** `plib.h`

**Code Example:**

```c
int chn=1; SpiChnRxTxIntDisable(chn);
mSpiChnRxTxIntDisable(chn);
```
SpiChnFaultIntEnable
mSpiChnFaultIntEnable

Description: This function/macro enables the SPI channel fault (overflow) interrupts.

Include: plib.h

Prototype: void SpiChnFaultIntEnable(int chn);

Arguments: chn This is the number of the SPI channel: 1 or 2

Remarks: Clears existing interrupt flags.

Return Value: None

Source File: plib.h

Code Example: int chn=1; SpiChnFaultIntEnable(chn);
mSpiChnFaultIntEnable(1);

SpiChnFaultIntDisable
mSpiChnFaultIntDisable

Description: This function/macro disables the SPI channel fault (overflow) interrupts.

Include: plib.h

Prototype: void SpiChnFaultIntDisable(int chn);

Arguments: chn This is the number of the SPI channel: 1 or 2

Remarks: Clears existing interrupt flags.

Return Value: None

Source File: plib.h

Code Example: int chn=1; SpiChnFaultIntDisable(chn);
mSpiChnFaultIntDisable(1);

14.5.5 INTERRUPT PRIORITY FUNCTIONS

SpiChnSetIntPriority
mSpiChnSetIntPriority

Description: This function/macro sets the SPI channel interrupt priority.

Include: plib.h

Prototype: void SpiChnSetIntPriority(int chn, int pri, int subPri);

Arguments: chn This is the number of the SPI channel: 1 or 2

pri the interrupt priority, 0 to 7

subPri the interrupt sub-priority, 0 to 3

Remarks: None

Return Value: None

Source File: plib.h
### SpiChnSetIntPriority
#### mSpiChnSetIntPriority

**Code Example:**
```c
int chn=1; int pri=5; int subPri=2;
SpiChnSetIntPriority(chn, pri, subPri);
mSpiChnSetIntPriority(1, pri, subPri);
```

### SpiChnGetIntPriority
#### mSpiChnGetIntPriority

**Description:** This function/macro returns the current SPI channel interrupt priority.

**Include:** `plib.h`

**Prototype:**
```c
int SpiChnGetIntPriority(int chn);
```

**Arguments:**
- `chn` This is the number of the SPI channel: 1 or 2

**Remarks:** None

**Return Value:** The current interrupt priority for the selected channel, 0 to 7

**Source File:** `plib.h`

**Code Example:**
```c
int chn=2; int currPri=SpiChnGetIntPriority(chn);
int currPri=mSpiChnGetIntPriority(2);
```

### SpiChnGetIntSubPriority
#### mSpiChnGetIntSubPriority

**Description:** This function/macro returns the current SPI channel interrupt sub-priority.

**Include:** `plib.h`

**Prototype:**
```c
int SpiChnGetIntSubPriority(int chn);
```

**Arguments:**
- `chn` This is the number of the SPI channel: 1 or 2

**Remarks:** None

**Return Value:** The current interrupt sub-priority for the selected channel, 0 to 3

**Source File:** `plib.h`

**Code Example:**
```c
int chn=2;
int currSPri=SpiChnGetIntSubPriority(chn);
int currSPri=mSpiChnGetIntSubPriority(2);
```
14.6 Example of Use

#include <plib.h>

// configuration settings
#pragma config POSCMOD = HS, FNOSC = PRIPLL
#pragma config PLLMUL = MUL_18, PLLIDIV = DIV_2
#pragma config FWDTEN = OFF

int main(void)
{
    // init the transmit buffer
    static const char txBuff[]="String of characters to be sent over the SPI channel";
    //room for the receive buffer
    static char rxBuff[sizeof(txBuff)];

    int ix;
    int rdData;
    const char*pSrc;
    char*pDst;
    int txferSize;
    int fail=0; // success flag

    // configure the proper PB frequency and the number of wait states
    SYSTEMConfigWaitStatesAndPB(72000000L);
    Checkseg0CacheOn(); // enable the cache for the best performance
    mBmXDisableDdrWaitState(); // no wait states for RAM

    // init the SPI chn 1 as master, 8 bits/character, frame master
    // divide fpb by 2
    SpiChnOpen(1, SPICON_MSTEN|SPICON_FRMEN|SPICON_SMP|SPICON_ON, 2);

    // init the SPI channel 2 as slave, 8 bits/character, frame slave
    // divide fpb by 2
    SpiChnOpen(2, SPICON_FRMEN|SPICON_FRMSYNC|SPICON_SMP|SPICON_ON, 2);

    txferSize=sizeof(txBuff);
    ix=txferSize+1;
    // transfer one extra word to give the slave the possibility
    // to reply back the last sent word
    pSrc=txBuff;
    pDst=rxBuff;

    while(ix--)
    {
        SpiChnPutC(1, *pSrc++); // send data on the master channel
        rdData=SpiChnGetC(1); // get the received data
        if(ix!=txferSize)
        { // skip the first received character, it's garbage
            *pDst++=rdData; // store the received data
        }
        rdData=SpiChnGetC(2); // receive data on the slave channel
        SpiChnPutC(2, rdData); // relay back data
    }

    // now let's check that the data was received ok
    pSrc=txBuff;
    pDst=rxBuff;
    for(ix=0; ix<sizeof(txBuff); ix++)
    {
        if(*pDst++!=*pSrc++)
        { fail=1; // data mismatch
        }
    }
}
break;
}
}
return !fail;
}
15.0 I²C™ FUNCTIONS

This section contains a list of individual functions for I²C module and an example of use of the functions. Functions may be implemented as macros.

15.1 Individual Functions

CloseI2C1
CloseI2C2

Description: This macro turns off the I²C module
Include: plib.h
Prototype: void CloseI2C1(void);
Arguments: None
Return Value None
Remarks: This function disables the I²C module and clears the Master and Slave Interrupt Enable and Flag bits.
Code Example: CloseI2C1();

AckI2C1
AckI2C2

Description: Generates I²C bus Acknowledge condition.
Include: plib.h
Prototype: void AckI2C1(void);
Arguments: None
Return Value None
Remarks: This function generates an I²C bus Acknowledge condition.
Code Example: AckI2C1();

DataRdyI2C1
DataRdyI2C2

Description: This macro provides status back to user if I2CxRCV register contain data.
Include: plib.h
Prototype: int DataRdyI2C1(void)
Arguments: None
Return Value This function returns '1' if there is data in I2CxRCV register; else return '0' which indicates no data in I2CxRCV register.
Remarks: This function determines if there is any byte to read from I2CxRCV register.
Code Example: if(!DataRdyI2C1());

IdleI2C1
IdleI2C2

Description: This function generates Wait condition until I²C bus is Idle.
Include: plib.h
Prototype: void IdleI2C1(void);
IdleI2C1
IdleI2C2 (Continued)

Arguments: None
Return Value None
Remarks: This function will be in a wait state until Start Condition Enable bit, Stop Condition Enable bit, Receive Enable bit, Acknowledge Sequence Enable bit of I2C Control register and Transmit Status bit I2C Status register are clear. The IdleI2C function is required since the hardware I2C peripheral does not allow for spooling of bus sequence. The I2C peripheral must be in Idle state before an I2C operation can be initiated or write collision will be generated.

Code Example: IdleI2C1();
**MastergetsI2C1**  
**MastergetsI2C2**

**Description:** This function reads predetermined data string length from the I²C bus.

**Include:**  
plib.h

**Prototype:**  
unsigned int MastergetsI2C1(unsigned int length, unsigned char *rdptr, unsigned int i2c_data_wait);

**Arguments:**
- **length**  Number of bytes to read from I²C device.
- **rdptr**  Character type pointer to RAM for storage of data read from I²C device.
- **i2c_data_wait**  This is the time-out count for which the module has to wait before return.  
  If the time-out count is 'N', the actual time out would be about (20 * N – 1) core cycles.

**Return Value**  
This function returns '0' if all bytes have been sent or number of bytes read from I²C bus if its not able to read the data within the specified i2c_data_wait time out value.

**Remarks:**  
This routine reads a predefined data string from the I²C bus.

**Code Example:**
```c
unsigned char string[10];  
unsigned char *rdptr;  
unsigned int length, i2c_data_wait;  
length = 9;  
rdptr = string;  
i2c_data_wait = 152;  
MastergetsI2C1(length, rdptr, i2c_data_wait);
```

---

**MasterputsI2C1**  
**MasterputsI2C2**

**Description:** This function is used to write out a data string to the I²C bus.

**Include:**  
plib.h

**Prototype:**  
unsigned int MasterputsI2C1(unsigned char *wrptr);

**Arguments:**  
- **wrptr**  Character type pointer to data objects in RAM. The data objects are written to the I²C device.

**Return Value**  
This function returns -3 if a write collision occurred. This function returns '0' if the null character was reached in data string.

**Remarks:**  
This function writes a string to the I²C bus until a null character is reached. Each byte is written via a call to the MasterputcI2C function. The actual called function body is termed MasterWriteI2C. MasterWriteI2C and MasterputcI2C refer to the same function via a #define statement in the plib.h

**Code Example:**
```c
unsigned char string[] = " MICROCHIP ";  
unsigned char *wrptr;  
wrptr = string;  
MasterputsI2C1( wrptr);
```
### MasterReadI2C1

**Description:** This function is used to read a single byte from I²C bus

**Include:** plib.h

**Prototype:**

```c
unsigned char MasterReadI2C1(void);
```

**Arguments:** None

**Return Value**
The return value is the data byte read from the I²C bus.

**Remarks:**
This function reads in a single byte from the I²C bus. This function performs the same function as `MastergetcI2C`.

**Code Example:**

```c
unsigned char value;
value = MasterReadI2C1();
```

### MasterWriteI2C1

**Description:** This function is used to write out a single data byte to the I²C device.

**Include:** plib.h

**Prototype:**

```c
unsigned char MasterWriteI2C1(unsigned char data_out);
```

**Arguments:**
- `data_out` A single data byte to be written to the I²C bus device.

**Return Value**
This function returns -1 if there was a write collision else it returns a 0.

**Remarks:**
This function writes out a single data byte to the I²C bus device. This function performs the same function as `MasterputcI2C`.

**Code Example:**

```c
MasterWriteI2C1('a');
```

### NotAckI2C1

**Description:** Generates I²C bus Not Acknowledge condition.

**Include:** plib.h

**Prototype:**

```c
void NotAckI2C1(void);
```

**Arguments:** None

**Return Value**
None

**Remarks:**
This function generates an I²C bus Not Acknowledge condition.

**Code Example:**

```c
NotAckI2C1();
```
OpenI2C1
OpenI2C2

Description: Configures the I2C module.
Include: plib.h
Prototype: void OpenI2C1(unsigned int config1, unsigned int brg);
Arguments: config1 This contains the parameter to configure the I2CCON register
I2C Enable bit
I2C_ON
I2C_OFF
I2C Stop in Idle Mode bit
I2C_IDLE_STOP
I2C_IDLE_CON
SCL Release Control bit
I2C_CLK_REL
I2C_CLK_HOLD
I2C Strict Addressing Mode
I2C_STRING_EN
I2C_STRING_DIS
10-bit Address bits
I2C_10BIT_ADD
I2C_7BIT_ADD
Slew Rate Control bit
I2C_SLW_DIS
I2C_SLW_EN
SMBus Input Level bits
I2C_SM_EN
I2C_SM_DIS
General Call Enable bit
I2C_GC_EN
I2C_GC_DIS
SCL Clock Stretch Enable bit
I2C_STR_EN
I2C_STR_DIS
Acknowledge Data bit
I2C_NACK (or I2C_ACKDT)
I2C_ACK
Acknowledge Sequence bit
I2C_ACK_EN
I2C_ACK_DIS
Receive Enable bit
I2C_RCV_EN
I2C_RCV_DIS
Stop Condition Enable bit
I2C_STOP_EN
I2C_STOP_DIS
Repeated Start Condition Enable bit
I2C_RESTART_EN
I2C_RESTART_DIS
Start Condition Enable bit
I2C_START_EN
I2C_START_DIS
### OpenI2C

**Description:**
- computed value for the baud rate generator. The value is calculated as follows: \( \text{BRG} = \frac{F_{pb}}{2 \times \text{baudrate}} - 2 \).

**Return Value:**
- None

**Remarks:**
- This function configures the I²C Control register and I²C Baud Rate Generator register.

**Code Example:**
```
OpenI2C1();
```

### RestartI2C

**Description:**
- Generates I²C Bus Restart condition.

**Include:**
- plib.h

**Prototype:**
- `void RestartI2C1(void);`

**Arguments:**
- None

**Return Value:**
- None

**Remarks:**
- This function generates an I²C Bus Restart condition.

**Code Example:**
```
RestartI2C1();
```

### SlavegetsI2C

**Description:**
- This function reads pre-determined data string length from the I²C bus.

**Include:**
- plib.h

**Prototype:**
- `unsigned int SlavegetsI2C1(unsigned char *rdptr, unsigned int i2c_data_wait);`

**Arguments:**
- `rdptr` Character type pointer to RAM for storage of data read from I²C device.
- `i2c_data_wait` This is the time-out count for which the module has to wait before return.

If the time-out count is 'N', the actual time out would be about \((20N - 1)\) core clock cycles.

**Return Value:**
- Returns the number of bytes received from the I²C bus.

**Remarks:**
- This routine reads a predefined data string from the I²C bus.

**Code Example:**
```
unsigned char string[12];
unsigned char *rdptr;
rdptr = string;
i2c_data_out = 0x11;
SlavegetsI2C1(rdptr, i2c_data_wait);
```
### SlaveputsI2C1

**Description:** This function is used to write out a data string to the I²C bus.

**Include:**
```
plib.h
```

**Prototype:**
```
unsigned int SlaveputsI2C1(unsigned char *wrptr);
```

**Arguments:**
- `wrptr` Character type pointer to data objects in RAM. The data objects are written to the I²C device.

**Return Value:**
This function returns '0' if the null character was reached in the data string.

**Remarks:**
This routine writes a data string out to the I²C bus until a null character is reached.

**Code Example:**
```
unsigned char string[] = "MICROCHIP";
unsigned char *rdptr;
rdptr = string;
SlaveputsI2C1(rdptr);
```

### SlaveReadI2C1

**Description:** This function is used to read a single byte from the I²C bus.

**Include:**
```
plib.h
```

**Prototype:**
```
unsigned char SlaveReadI2C1(void);
```

**Arguments:** None

**Return Value:** The return value is the data byte read from the I²C bus.

**Remarks:**
This function reads in a single byte from the I²C bus. This function performs the same function as SlavegetcI2C.

**Code Example:**
```
unsigned char value;
value = SlaveReadI2C1();
```

### SlaveWriteI2C1

**Description:** This function is used to write out a single byte to the I²C bus.

**Include:**
```
plib.h
```

**Prototype:**
```
void SlaveWriteI2C2(unsigned char data_out);
```

**Arguments:**
- `data_out` A single data byte to be written to the I²C bus device.

**Return Value:** None

**Remarks:**
This function writes out a single data byte to the I²C bus device. This function performs the same function as SlaveputcI2C.

**Code Example:**
```
SlaveWriteI2C2('a');
```
## 15.2 Individual Macros

### StartI2C1

**Description:** Generates I²C Bus Start condition.

**Include:** 
plib.h

**Prototype:**

```c
void StartI2C1(void);
```

**Arguments:**

None

**Return Value:** None

**Remarks:**
This function generates a I²C Bus Start condition.

**Code Example:**

```c
StartI2C1();
```

### StopI2C1

**Description:** Generates I²C Bus Stop condition.

**Include:** 
plib.h

**Prototype:**

```c
void StopI2C1(void);
```

**Arguments:**

None

**Return Value:** None

**Remarks:**
This function generates a I²C Bus Stop condition.

**Code Example:**

```c
StopI2C1();
```

### EnableIntMI2C1

**Description:**
This macro enables the master I²C interrupt.

**Include:** 
plib.h

**Arguments:**

None

**Remarks:**
This macro sets Master I²C Enable bit of Interrupt Enable Control register.

**Code Example:**

```c
EnableIntMI2C1;
```

### DisableIntMI2C1

**Description:**
This macro disables the master I²C interrupt.

**Include:** 
plib.h

**Arguments:**

None

**Remarks:**
This macro clears Master I²C Interrupt Enable bit of Interrupt Enable Control register.

**Code Example:**

```c
DisableIntMI2C1;
```

### EnableIntBI2Cx

**Description:**
This macro enables or disables the bus collision I²C interrupt.

**Include:** 
plib.h

**Arguments:**

None

**Remarks:**

```c
EnableIntBI2Cx;
```
EnableIntBI2Cx
DisableIntBI2Cx

Include: plib.h
Arguments: None
Remarks: This macro sets or clears Bus Collision I²C Interrupt Enable bit of Interrupt Enable Control register.
Code Example: DisableIntBI2C1;
SetPriorityIntI2C1
SetPriorityIntI2C2

Description: This macro sets priority for I2C interrupt.
Include: plib.h
Prototype: void SetPriorityIntI2C1(unsigned int config);
Arguments: config This contains the bit fields that make up the parameter. A logical OR is used to combine multiple bit fields together.

<table>
<thead>
<tr>
<th>Interrupt priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C1_INT_PRI_0 or I2C2_INT_PRI_0</td>
</tr>
<tr>
<td>I2C1_INT_PRI_1 or I2C2_INT_PRI_1</td>
</tr>
<tr>
<td>I2C1_INT_PRI_2 or I2C2_INT_PRI_2</td>
</tr>
<tr>
<td>I2C1_INT_PRI_3 or I2C2_INT_PRI_3</td>
</tr>
<tr>
<td>I2C1_INT_PRI_4 or I2C2_INT_PRI_4</td>
</tr>
<tr>
<td>I2C1_INT_PRI_5 or I2C2_INT_PRI_5</td>
</tr>
<tr>
<td>I2C1_INT_PRI_6 or I2C2_INT_PRI_6</td>
</tr>
<tr>
<td>I2C1_INT_PRI_7 or I2C2_INT_PRI_7</td>
</tr>
</tbody>
</table>
(These bit fields are mutually exclusive)

<table>
<thead>
<tr>
<th>Interrupt sub priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C1_SUB_INT_PRI_0 or I2C2_SUB_INT_PRI_0</td>
</tr>
<tr>
<td>I2C1_SUB_INT_PRI_1 or I2C2_SUB_INT_PRI_1</td>
</tr>
<tr>
<td>I2C1_SUB_INT_PRI_2 or I2C2_SUB_INT_PRI_2</td>
</tr>
<tr>
<td>I2C1_SUB_INT_PRI_3 or I2C2_SUB_INT_PRI_3</td>
</tr>
</tbody>
</table>
(These bit fields are mutually exclusive)

Remarks: This macro sets I2C Interrupt Priority bits of Interrupt Priority Control register.
Code Example: SetPriorityIntI2C1(I2C1_INT_PRI_2 | I2C1_INT_SUB_PRI_3);

EnableIntSI2C1
EnableIntSI2C2

Description: This macro enables the slave I2C interrupt.
Include: plib.h
Arguments: None
Remarks: This macro sets Slave I2C Enable bit of Interrupt Enable Control register.
Code Example: EnableIntSI2C1;

DisableIntSI2C1
DisableIntSI2C2

Description: This macro disables the slave I2C interrupt.
Include: plib.h
Arguments: None
Remarks: This macro clears Slave I2C Interrupt Enable bit of Interrupt Enable Control register.
Code Example: DisableIntSI2C1;
15.3 Example of Use

#include <plib.h>

// Configuration Bit settings
// System Clock = 60 MHz, Peripheral Bus = 7.5MHz
// Primary Osc w/PLL (XT+, HS+, EC+PLL)
// Input Divider 2x Divider
// Multiplier 15x Multiplier

#define CCLK(60000000) //8Mhz Osc on Explorer16 board (pll 8 / 2 * 15)
#define PBCLK (CCLK/8)
#define Fsck375000
#define BRG_VAL (PBCLK/2/Fsck)
#define Nop() asm( "nop" )

int main(void)
{
    unsigned char SlaveAddress;
    char i2cData[10];
    int DataSz;

    // Set Periph Bus Divider 60MHz / 8 = 9MHz Fpb
    mOSCSetPBDIV( OSC_PB_DIV_8 );

    //Enable channel
    OpenI2C1( I2C_EN, BRG_VAL );

    SlaveAddress = 0x50;//0b1010000 Serial EEPROM address
// Send Data to eeprom to program one location

i2cData[0] = (SlaveAddress << 1) | 0;//EEPROM Device Address and WR Command
i2cData[1] = 0x05;//eeprom location to program (high address byte)
i2cData[2] = 0x40;//eeprom location to program (low address byte)
i2cData[3] = 0xAA;//data to write
DataSz = 4;

StartI2C1();//Send the Start Bit
IdleI2C1();//Wait to complete

int Index = 0;
while( DataSz )
{
    MasterWriteI2C1( i2cData[Index++] );
    IdleI2C1();//Wait to complete
    DataSz--;
}

//ACKSTAT is 0 when slave acknowledge. if 1 then slave has not acknowledge the data.
if( I2C1STATbits.ACKSTAT )
    break;
}

StopI2C1();//Send the Stop condition
IdleI2C1();//Wait to complete

// wait for eeprom to complete write process. poll the ack status
while(1)
{
    i2c_wait(10);
}

StartI2C1();//Send the Start Bit
IdleI2C1();//Wait to complete

MasterWriteI2C1( i2cData[0] );
IdleI2C1();//Wait to complete

if( I2C1STATbits.ACKSTAT == 0 }//eeprom has acknowledged
{
    StopI2C1();//Send the Stop condition
    IdleI2C1();//Wait to complete
break;
}

StopI2C1();//Send the Stop condition
IdleI2C1();//Wait to complete
}

// Now Readback the data from the serial eeprom

i2cData[0] = (SlaveAddress << 1) | 0;//EEPROM Device Address and WR Command (to write the address)
i2cData[1] = 0x05;//EEPROM location to read (high address byte)
i2cData[2] = 0x40;//EEPROM location to read (low address byte)
DataSz = 3;

StartI2C1();//Send the Start Bit
IdleI2C1();//Wait to complete

// Send the address to read from the serial eeprom
Index = 0;
while( DataSz )
{
    MasterWriteI2C1( i2cData[Index++] );
    IdleI2C1();//Wait to complete
    DataSz--;

    // ACKSTAT is 0 when slave acknowledge. if 1 then slave has not acknowledge the data.
    if( I2C1STATbits.ACKSTAT )
        break;
}

// now send a start sequence again
RestartI2C1();//Send the Stop condition
IdleI2C1();//Wait to complete

MasterWriteI2C1( (SlaveAddress << 1) | 1 ); // transmit read command
IdleI2C1();//Wait to complete

unsigned char i2cbyte;
i2cbyte = MasterReadI2C1();

StopI2C1();//Send the Stop condition
IdleI2C1();//Wait to complete
if( i2cbyte != 0xAA )
{
    while(1) //error: verify failed
    {}
}

while(1) // Success
{}
}
16.0 UART FUNCTIONS

This section contains a list of individual functions for UART module and an example of use of the functions. Functions may be implemented as macros.

16.1 Individual Functions

**BusyUART1**
**BusyUART2**

| Description: | This macro returns the UART transmission status. |
| Include: | plib.h |
| Prototype: | int BusyUART1(void); int BusyUART2(void); |
| Arguments: | None |
| Return Value: | If Non-Zero value is returned, it indicates that UART is busy in transmission and UxSTA<TRMT> bit is ‘0’. If ‘0’ is returned, it indicates that UART is not busy and UxSTA<TRMT> bit is ‘1’. |
| Remarks: | This macro returns the status of the UART. This indicates if the UART is busy in transmission as indicated by the UxSTA<TRMT> bit. |
| Code Example: | while(BusyUART1()); |

**CloseUART1**
**CloseUART2**

| Description: | This macro turns off the UART module |
| Include: | plib.h |
| Prototype: | void CloseUART1(void); void CloseUART2(void); |
| Arguments: | None |
| Return Value: | None |
| Remarks: | This macro first turns off the UART module and then disables the UART transmit and receive interrupts. The Interrupt Flag bits are also cleared. |
| Code Example: | CloseUART1(); |
ConfigIntUART1
ConfigIntUART2

Description: This macro configures the UART interrupts.

Include: pplib.h

Prototype:

void ConfigIntUART1(unsigned int config);
void ConfigIntUART2(unsigned int config);

Arguments: config

Individual interrupt enable/disable information as defined below:

**Error Interrupt enable**

- UART_ERR_INT_EN
- UART_ERR_INT_DIS

(These bit fields are mutually exclusive)

**Receive Interrupt enable**

- UART_RX_INT_EN
- UART_RX_INT_DIS

(These bit fields are mutually exclusive)

**UART Interrupt Priority**

- UART_INT_PR0
- UART_INT_PR1
- UART_INT_PR2
- UART_INT_PR3
- UART_INT_PR4
- UART_INT_PR5
- UART_INT_PR6
- UART_INT_PR7

(These bit fields are mutually exclusive)

**UART Interrupt Sub-Priority**

- UART_INT_SUB_PR0
- UART_INT_SUB_PR1
- UART_INT_SUB_PR2
- UART_INT_SUB_PR3

(These bit fields are mutually exclusive)

**Transmit Interrupt enable**

- UART_TX_INT_EN
- UART_TX_INT_DIS

(These bit fields are mutually exclusive)

**Transmit Interrupt Priority**

- UART_TX_INT_PR0
- UART_TX_INT_PR1
- UART_TX_INT_PR2
- UART_TX_INT_PR3
- UART_TX_INT_PR4
- UART_TX_INT_PR5
- UART_TX_INT_PR6
- UART_TX_INT_PR7

(These bit fields are mutually exclusive)

Return Value: None

Remarks: This macro enables/disables the UART transmit and receive interrupts and sets the interrupt priorities.

Code Example:

ConfigIntUART1(UART_RX_INT_EN | UART_TX_INT_DIS |
UART_ERR_INT_EN | UART_INT_PR0 | UART_INT_SUB_PR0);
### DataRdyUART1
### DataRdyUART2

**Description:** This macro returns the UART receive buffer status.

**Include:** `plib.h`

**Prototype:**
```c
int DataRdyUART1(void);
int DataRdyUART2(void);
```

**Arguments:** None

**Return Value:** If Non-Zero value is returned, it indicates that the receive buffer has a data to be read. If `0` is returned, it indicates that receive buffer does not have any new data to be read.

**Remarks:** This macro returns the status of the UART receive buffer. This indicates if the UART receive buffer contains any new data that is yet to be read as indicated by the UxSTA<URXDA> bit.

**Code Example:**
```c
while(DataRdyUART1());
```

### OpenUART1
### OpenUART2

**Description:** This macro configures the UART module

**Include:** `plib.h`

**Prototype:**
```c
void OpenUART1(unsigned int config1,
                unsigned int config2,
                unsigned int ubrg);
void OpenUART2(unsigned int config1,
                unsigned int config2,
                unsigned int ubrg);
```

**Arguments:**
- **config1**
  - This contains the parameters to be configured in the UxMODE register as defined below:
    - **UART enable/disable**
      - UART_EN
      - UART_DIS
      (These bit fields are mutually exclusive)
    - **UART Idle mode operation**
      - UART_IDLE_CON
      - UART_IDLE_STOP
      (These bit fields are mutually exclusive)
    - **UART communication with ALT pins**
      - UART_ALTRX_ALTTX
      - UART_RX_TX
      (These bit fields are mutually exclusive)
    - **UART Wake-up on Start**
      - UART_EN_WAKE
      - UART_DIS_WAKE
      (These bit fields are mutually exclusive)
    - **UART Loopback mode enable/disable**
      - UART_EN_LOOPBACK
      - UART_DIS_LOOPBACK
      (These bit fields are mutually exclusive)
    - **Input to Capture module**
      - UART_EN_ABAUD
      - UART_DIS_ABAUD
      (These bit fields are mutually exclusive)
### OpenUART1 (Continued)

#### UART Mode Select bits
- UART_EN_BCLK
- UART_EN_CTS_RTS
- UART_EN_RTS
- UART_DIS_BCLK_CTS_RTS

#### UART Transmission mode interrupt flag select
- UART_INT_TX_BUF_EMPTY
- UART_INT_TX_LAST_CH
- UART_INT_TX

#### UART Transmit Break bit
- UART_TX_PIN_NORMAL
- UART_TX_PIN_LOW

#### UART transmit enable/disable
- UART_TX_ENABLE
- UART_TX_DISABLE

#### UART recieve enable/disable
- UART_RX_ENABLE
- UART_RX_DISABLE

---

**Parity and data bits select**
- UART_NO_PAR_9BIT
- UART_ODD_PAR_8BIT
- UART_EVEN_PAR_8BIT
- UART_NO_PAR_8BIT

(These bit fields are mutually exclusive)

**Number of Stop bits**
- UART_2STOPBITS
- UART_1STOPBIT

(These bit fields are mutually exclusive)

**IRDA Enable/Disable**
- UART_IRDA_EN
- UART_IRDA_DIS

(These bit fields are mutually exclusive)

**RTS Mode Select**
- UART_MODE_SIMPLEX
- UART_MODE_FLOWCTRL

(These bit fields are mutually exclusive)

**Recievie Polarity**
- UART_INVERT_RX
- UART_NORMAL_RX

(These bit fields are mutually exclusive)

**High Baud Rate Select**
- UART_BRGH_FOUR
- UART_BRGH_SIXTEEN

(These bit fields are mutually exclusive)

---

`config2` This contains the parameters to be configured in the UxSTA register as defined below:
### OpenUART1 (Continued)
### OpenUART2

**UART Receive Interrupt mode select**
- UART_INT_RX_BUF_FUL
- UART_INT_RX_3_4_FUL
- UART_INT_RX_CHAR

(These bit fields are mutually exclusive)

**UART address detect enable/disable**
- UART_ADR_DETECT_EN
- UART_ADR_DETECT_DIS

(These bit fields are mutually exclusive)

**UART OVERRUN bit clear**
- UART_RX_OVERRUN_CLEAR

(These bit fields are mutually exclusive)

| ubrg | This is the value to be written into UxBRG register to set the baud rate. |

**Return Value:** None  
**Remarks:** This macro configures the UART transmit and receive sections and sets the communication baud rate.  
**Code Example:**  
```
OpenUART1(UART_EN | UART_BRGH_FOUR,  
          UART_TX_PIN_NORMAL | UART_RX_EN | UART_TX_ENABLE,  
          123);
```
### ReadUART1
### ReadUART2

**Description:** This macro returns the content of UART receive buffer (UxRXREG) register.

**Include:** plib.h

**Prototype:**
- unsigned int ReadUART1(void);
- unsigned int ReadUART2(void);

**Arguments:** None

**Return Value:** This macro returns the contents of Receive buffer (UxRXREG) register.

**Remarks:**
- This macro returns the contents of the Receive Buffer register.
- If 9 bit reception is enabled, the entire register content is returned.
- If 8 bit reception is enabled, then register is read and the 9th bit is masked.

**Code Example:**
```c
unsigned int RX_data;
RX_data = ReadUART1();
```

### WriteUART1
### WriteUART2

**Description:** This macro writes data to be transmitted into the transmit buffer (UxTXREG) register.

**Include:** plib.h

**Prototype:**
- void WriteUART1(unsigned int data);
- void WriteUART2(unsigned int data);

**Arguments:**
- data This is the data to be transmitted.

**Return Value:** None

**Remarks:**
- This macro writes the data to be transmitted into the transmit buffer.
- If 9-bit transmission is enabled, the 9-bit value is written into the transmit buffer.
- If 8-bit transmission is enabled, then upper byte is masked and then written into the transmit buffer.

**Code Example:**
```c
WriteUART1('a');
```
getsUART1
getsUART2

Description: This function reads a string of data of specified length and stores it into the buffer location specified.

Include: plib.h

Prototype:
unsigned int getsUART1(unsigned int length, char *buffer, unsigned int uart_data_wait);
unsigned int getsUART2(unsigned int length, unsigned int *buffer, unsigned int uart_data_wait);

Arguments:
length This is the length of the string to be received.
buffer This is the pointer to the location where the data received have to be stored.
uart_data_wait This is the time-out count for which the module has to wait before return.
If the time-out count is \( N \), the actual time out would be about \( (19 \times N - 1) \) instruction cycles.

Return Value: This function returns the number of bytes yet to be received.
If the return value is '0', it indicates that the complete string has been received.
If the return value is non-zero, it indicates that the complete string has not been received.

Remarks: None

Code Example: getsUART1(12, myBuffer, 123);

putsUART1
putsUART2

Description: This function writes a string of data to be transmitted into the UART transmit buffer.

Include: plib.h

Prototype:
void putsUART1(const char *buffer);
void putsUART2(const char *buffer);

Arguments: buffer This is the pointer to the string of data to be transmitted.

Return Value: None

Remarks: This function writes the data to be transmitted into the transmit buffer until NULL character is encountered.
Once the transmit buffer is full, it waits until data gets transmitted and then writes the next data into the Transmit register.

Code Example: putsUART1(“Hello World!”);
### getcUART1
### getcUART2

**Description:**
This macro is identical to ReadUART1 and ReadUART2.

### putcUART1
### putcUART2

**Description:**
This macro is identical to WriteUART1 and WriteUART2.

### UART1GetErrors
### UART2GetErrors

**Description:**
This macro retrieves bitmap of various error values.

**Include:**
plib.h

**Prototype:**
```c
int UART1GetErrors (void);
int UART2GetErrors (void);
```

**Arguments:**
None.

**Return Value:**
- bit b0 : '1' Overflow error, '0' - No overflow error
- b1 : '1' Frame error, '0' - No frame error
- b2 : '1' Parity error, '0' - No parity error

**Remarks:**

**Code Example:**
```c
errorValue = UART1GetErrors();
if (errorValue & 0x01) // Overflow error
if (errorValue & 0x02) // Frame error
if (errorValue & 0x04) // Parity error
```

### UART1ClearErrors
### UART2ClearErrors

**Description:**
This macro clears all error flags.

**Include:**
plib.h

**Prototype:**
```c
void UART1ClearErrors(void);
void UART2ClearErrors(void);
```

**Arguments:**
None.

**Return Value:**
None.

**Remarks:**

**Code Example:**
```c
UART1ClearErrors();
```
## 16.2 Individual Macros

### EnableIntU1RX  
**Description:** This macro enables the UART receive interrupt.  
**Include:** plib.h  
**Arguments:** None  
**Remarks:** This macro sets UART Receive Interrupt Enable bit of Interrupt Enable Control register.  
**Code Example:** EnableIntU2RX;

### EnableIntU1TX  
**Description:** This macro enables the UART transmit interrupt.  
**Include:** plib.h  
**Arguments:** None  
**Remarks:** This macro sets UART Transmit Interrupt Enable bit of Interrupt Enable Control register.  
**Code Example:** EnableIntU2TX;

### DisableIntU1RX  
**Description:** This macro disables the UART receive interrupt.  
**Include:** plib.h  
**Arguments:** None  
**Remarks:** This macro clears UART Receive Interrupt Enable bit of Interrupt Enable Control register.  
**Code Example:** DisableIntU1RX;
## DisableIntU1TX

**Description:**
This macro disables the UART transmit interrupt.

**Include:**
plib.h

**Arguments:**
None

**Remarks:**
This macro clears UART Transmit Interrupt Enable bit of Interrupt Enable Control register.

**Code Example:**
DisableIntU1TX;

## SetPriorityIntU1

**Description:**
This macro sets priority for UART channel.

**Include:**
plib.h

**Arguments:**
priority

<table>
<thead>
<tr>
<th>Priority Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_INT_PR0</td>
</tr>
<tr>
<td>UART_INT_PR1</td>
</tr>
<tr>
<td>UART_INT_PR2</td>
</tr>
<tr>
<td>UART_INT_PR3</td>
</tr>
<tr>
<td>UART_INT_PR4</td>
</tr>
<tr>
<td>UART_INT_PR5</td>
</tr>
<tr>
<td>UART_INT_PR6</td>
</tr>
<tr>
<td>UART_INT_PR7</td>
</tr>
</tbody>
</table>

**Remarks:**
This macro sets UART Interrupt Priority bits of Interrupt Priority Control register.

**Code Example:**
SetPriorityIntU1(UART_INT_PR3);

## SetSubPriorityIntU1

**Description:**
This macro sets the sub priority for UART channel.

**Include:**
plib.h

**Arguments:**
sub_priority

<table>
<thead>
<tr>
<th>Sub Priority Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_INT_SUB_PR0</td>
</tr>
<tr>
<td>UART_INT_SUB_PR1</td>
</tr>
<tr>
<td>UART_INT_SUB_PR2</td>
</tr>
<tr>
<td>UART_INT_SUB_PR3</td>
</tr>
</tbody>
</table>

**Remarks:**
This macro sets UART Interrupt Sub Priority bits of Interrupt Priority Control register.

**Code Example:**
SetSubPriorityIntU1(UART_INT_SUB_PR3);
16.3 Example of Use

**UART1SendBreak**

**UART2SendBreak**

**Description:** This macro Initiates Break sequence on UARTx.

**Include:** plib.h

**Prototype:**

```c
void UART1SendBreak(void);
void UART2SendBreak(void);
```

**Arguments:** None.

**Return Value:** None.

**Remarks:**

**Code Example:**

```c
UART1SendBreak();
```

---

**UART1EnableAutoAddr**

**UART2EnableAutoAddr**

**Description:** This macro Enables the automatic address matching mode of UART.

**Include:** plib.h

**Prototype:**

```c
void UART1EnableAutoAddr(int address);
void UART2EnableAutoAddr(int address);
```

**Arguments:**

- `address`: The 9-bit address for this UART.

**Return Value:** None.

**Remarks:**

**Code Example:**

```c
UART1EnableAutoAddr(0x18);
```
17.0 PMP FUNCTIONS

The PIC32MX PMP library consists of functions and macros supporting common configuration and control features.

• **PMP Common Operations**
  - mPMPOpen
  - mPMPClose
  - mPMPEnable
  - mPMPDisable
  - mPMPIdleStop
  - mPMPIdleContinue

• **PMP Master Port Operations**
  - PMPSetAddress
  - PMPMasterRead
  - mPMPMasterReadByte
  - mPMPMasterReadWord
  - PMPMasterReadByteBlock
  - PMPMasterReadWordBlock
  - PMPMasterWrite
  - PMPMasterWriteByteBlock
  - PMPMasterWriteWordBlock
  - mIsPMPBusy
  - mPMPGetBusyFlag

• **PMP Slave Port Operations**
  - mPMPSlaveRead
  - PMPSlaveReadBuffer
  - PMPSlaveReadBuffers
  - mPMPSlaveWrite
  - PMPSlaveWriteBuffer
  - PMPSlaveWriteBuffers
  - mPMPGetBufferFullFlags
  - mIsPMPSlaveBufferFull
  - mPMPGetBufferEmptyFlags
  - mIsPMPSlaveBufferEmpty
  - mIsPMPSlaveBufferOverflow
  - mPMPClearBufferOverflow
  - mIsPMPSlaveBufferUnderflow
  - mPMPClearBufferUnderflow


17.1 Common Functions and Macros

mPMPOpen

Description: This macro configures the PMP module.

Include: plib.h

Prototype: void mPMPOpen(unsigned int ctrl, unsigned int mode,
unsigned int port, unsigned int intr);

Arguments: ctrl PMP control configuration. This argument contains one or
more bit masks bitwise OR'd together. Select one or more
from the following bit masks. Note: An absent mask symbol
in an argument assumes the corresponding bit(s) are
disabled, or default value, and are set = 0.

PMP module On/Off
PMP_ON
PMP_OFF
(These bit fields are mutually exclusive)

PMP idle mode On/Off
PMP_IDLE_CON
PMP_IDLE_STOP
(These bit fields are mutually exclusive)

PMP address multiplex mode
PMP_MUX_DATA16_ALL
PMP_MUX_DATA8_ALL
PMP_MUX_DATA8_LOWER
PMP_MUX_OFF
(These bit fields are mutually exclusive)

PMP read and write strobe enable
PMP_READ_WRITE_EN
PMP_WRITE_EN
PMP_READ_EN
PMP_READ_WRITE_OFF
(These bit fields are mutually exclusive)

PMP Input Buffer Type Select
PMP_TTL
PMP_ST
(These bit fields are mutually exclusive)

PMP chip select function
PMP_CS2_CS1_EN
PMP_CS2_EN
PMP_CS2_CS1_OFF
(These bit fields are mutually exclusive)

PMP address latch polarity
PMP_LATCH_POL_HI
PMP_LATCH_POL_LO
(These bit fields are mutually exclusive)

PMP chip select polarity
PMP_CS2_POL_HI
PMP_CS2_POL_LO
PMP_CS1_POL_HI
PMP_CS1_POL_LO
(These bit fields are mutually exclusive)
mPMPOpen (Continued)

| PMP read and write polarity          | PMP_WRITE_POL_HI       | PMP_WRITE_POL_LO       |
|                                      | PMP_READ_POL_HI       | PMP_READ_POL_LO       |

(These bit fields are mutually exclusive)

PMP mode configuration. This argument contains one or more bit masks bitwise OR’d together. Select one or more from the following bit mask. Note: An absent mask symbol in an argument assumes the corresponding bit(s) are disabled, or default value, and are set = 0.

| PMP interrupt request mode           | PMP_IRQ_BUF_FULL       |
|                                      | PMP_IRQ_READ_WRITE     |
|                                      | PMP_IRQ_OFF            |

(These bit fields are mutually exclusive)

| PMP address increment mode           | PMP_AUTO_ADDR_BUFFER   |
|                                      | PMP_AUTO_ADDR_DEC      |
|                                      | PMP_AUTO_ADDR_INC      |
|                                      | PMP_AUTO_ADDR_OFF      |

(These bit fields are mutually exclusive)

| PMP data width                       | PMP_DATA_BUS_8         |
|                                      | PMP_DATA_BUS_16        |

(These bit fields are mutually exclusive)

| PMP module mode                     | PMP_MODE_MASTER_1      |
|                                      | PMP_MODE_MASTER_2      |
|                                      | PMP_MODE_ESLAVE        |
|                                      | PMP_MODE_SLAVE        |

(These bit fields are mutually exclusive)

| PMP beginning phase wait cycles     | PMP_WAIT_BEG_4         |
|                                      | PMP_WAIT_BEG_3         |
|                                      | PMP_WAIT_BEG_2         |
|                                      | PMP_WAIT_BEG_1         |

(These bit fields are mutually exclusive)
mPMPOpen (Continued)

PMP middle phase wait cycles
PMP_WAIT_MID_15
PMP_WAIT_MID_14
PMP_WAIT_MID_13
PMP_WAIT_MID_12
PMP_WAIT_MID_11
PMP_WAIT_MID_10
PMP_WAIT_MID_9
PMP_WAIT_MID_8
PMP_WAIT_MID_7
PMP_WAIT_MID_6
PMP_WAIT_MID_5
PMP_WAIT_MID_4
PMP_WAIT_MID_3
PMP_WAIT_MID_2
PMP_WAIT_MID_1
PMP_WAIT_MID_0
(These bit fields are mutually exclusive)

PMP end phase wait cycles
PMP_WAIT_END_4
PMP_WAIT_END_3
PMP_WAIT_END_2
PMP_WAIT_END_1
(These bit fields are mutually exclusive)

port  PMP port pin configuration. This argument contains one or more bit masks bitwise OR’d together. Select one or more from the following bit mask. Note: An absent mask symbol in an argument assumes the corresponding bit(s) are disabled, or default value, and are set = 0.

PMP port pin enable
PMP_PEN_ALL
PMP_PEN_15
PMP_PEN_14
PMP_PEN_13
PMP_PEN_12
PMP_PEN_11
PMP_PEN_10
PMP_PEN_9
PMP_PEN_8
PMP_PEN_7
PMP_PEN_6
PMP_PEN_5
PMP_PEN_4
PMP_PEN_3
PMP_PEN_2
PMP_PEN_1
PMP_PEN_0
PMP_PEN_OFF
(These bit fields are mutually exclusive)

intr  PMP interrupt configuration. This argument contains one or more bit masks bitwise OR’d together. Select one or more from the following bit mask. Note: An absent mask symbol in an argument assumes the corresponding bit(s) are disabled, or default value, and are set = 0.
mPMPOpen (Continued)

PMP interrupt on/off
PMP_INT_ON
PMP_INT_OFF
(These bit fields are mutually exclusive)

PMP interrupt priorities
PMP_INT_PRI_7
PMP_INT_PRI_6
PMP_INT_PRI_5
PMP_INT_PRI_4
PMP_INT_PRI_3
PMP_INT_PRI_2
PMP_INT_PRI_1
PMP_INT_PRI_0
(These bit fields are mutually exclusive)

Return Value: None.
Remarks: This function clears PMP interrupt flag, configures the PMP module and interrupt priority then enables the module.

Code Example:
/* Open PMP module using master mode 2 */
#define CONTROL  (PMP_ON | PMP_IDLE_CON |
                  PMP_MUX_DATA8_LOWER |
                  PMP_READ_WRITE_EN |
                  PMP_CS2_CS1_EN |
                  PMP_LATCH_POL_HI |
                  PMP_CS2_POL_LO | PMP_CS1_POL_LO |
                  PMP_WRITE_POL_LO | PMP_READ_POL_LO)
#define MODE     (PMP_IRQ_OFF | PMP_AUTO_ADDR_OFF |
                  PMP_DATA_BUS_8 | PMP_MODE_MASTER2 |
                  PMP_WAIT_BEG_3 | PMP_WAIT_MID_7 |
                  PMP_WAIT_END_3 )
#define PORT     (PMP_PEN_ALL)
#define INT      (PMP_INT_ON | PMP_INT_PRI_4)

mPMPOpen(CONTROL, MODE, PORT, INT);
### mPMPClose
- **Description:** This macro turns the PMP module off.
- **Include:** `plib.h`
- **Prototype:**

```c
void mPMPClose(void);
```
- **Arguments:** None
- **Return Value:** None
- **Remarks:** This function clears the PMP interrupt flag, disables the PMP module and interrupt.
- **Code Example:**

```c
mPMPClose();
```

### mPMPEnable
- **Description:** This macro enables the PMP module
- **Include:** `plib.h`
- **Prototype:**

```c
void mPMPEnable(void);
```
- **Arguments:** None
- **Return Value:** None
- **Remarks:** This macro sets bit PMCON<ON> = 1
- **Code Example:**

```c
mPMPEnable();
```

### mPMPDisable
- **Description:** This macro disables the PMP module
- **Include:** `plib.h`
- **Prototype:**

```c
void mPMPDisable(void);
```
- **Arguments:** None
- **Return Value:** None
- **Remarks:** This macro sets bit PMCON<ON> = 0
- **Code Example:**

```c
mPMPDisable();
```
mPMPIdleStop

Description: This macro configures the PMP to stop operating when cpu enters idle mode

Include: plib.h
Prototype: void mPMPIdleStop(void);

Arguments: None
Return Value: None
Remarks: Code Example: mPMPIdleStop();

mPMPIdleContinue

Description: This macro configures the PMP to continue operating when cpu enters idle mode

Include: plib.h
Prototype: void mPMPIdleContinue(void);

Arguments: None
Return Value: None
Remarks: Code Example: mPMPIdleContinue();

17.2 Master Mode Functions and Macros

PMPSetAddress

Description: This function sets the address that will appear on the PMP bus when a master read or write operation is performed.

Include: plib.h
Prototype: void PMPSetAddress(unsigned int address)

Arguments: address A value in the range 0x0000 - 0xFFFF
Return Value: None
Remarks: This function polls the PMP Busy flag to ensure any previous read or write operation has completed prior to updating the PMADDRS register.

Code Example: void PMPSetAddress(0x4200);
PMPMasterRead

Description: This function returns data read from an external device connected to
the PMP port.

Include: plib.h

Prototype: unsigned int PMPMasterRead(void);

Arguments: None

Return Value: The latched value from the previous bus read.

Remarks: This function polls the PMP Busy flag to ensure any previous read or
write operation has completed prior to reading the PMDIN register. Note that the read data obtained from the PMDIN register is actually the
value latched from the previous read operation. Hence, the first user
read will be a dummy read to initiate the first bus read and fill the read
register.

Depending on the PMP mode, the data could be 8-bit or 16-bit, however, the value returned is always 32-bits wide. For example, in 8-
bit mode, a value of 0xFF read from an external device will be returned as 0x000000FF. Likewise, in 16-bit mode, a value of 0xFFFF read from
an external device will be returned as 0x0000FFFF.

See mPMPMasterReadByte and mPMPMasterReadWord macros below if casting the return value to a specific size is required.

Code Example:

/* example */
unsigned int ReadValue;
ReadValue = PMPMasterRead();

/* example using casting */
unsigned char ReadValue8;
unsigned short ReadValue16;
ReadValue8 = (unsigned char) PMPMasterRead();
ReadValue16 = (unsigned short) PMPMasterRead();

mPMPMasterReadByte

Description: This macro calls PMPMasterRead

Include: plib.h

Prototype: unsigned char mPMPMasterReadByte(void);

Arguments: None

Return Value: unsigned char

Remarks: This macro calls PMPMasterRead() and casts the return value =
unsigned char.

Code Example:

/* example using function in 8-bit PMP mode */
unsigned char ReadValue8;
ReadValue8 = mPMPMasterReadByte();
mPMPMasterReadWord

Description: This macro enables the PMP module
Include: plib.h
Prototype: unsigned short mPMPMasterReadWord(void);
Arguments: None
Return Value: unsigned short
Remarks: This macro calls PMPMasterRead() and casts the return value = unsigned short.

Code Example: /* example using function in 16-bit PMP mode */
unsigned short ReadValue16;
ReadValue16 = mPMPMasterReadWord();

PMPMasterReadByteBlock

Description: This function reads a block of 8-bit (byte) data from an external device.
Include: plib.h
Prototype: void PMPMasterReadByteBlock(unsigned int address, unsigned int bytes, unsigned char* pDest);
Arguments: address External 16-bit starting address.
bytes The number of bytes to read.
pDest 8-bit (byte) pointer to user memory where the data will be copied.
Return Value: None
Remarks: This function polls the PMP Busy flag to ensure any previous read or write operation has completed prior to reading a block of data. This function is intended for use when the PMP is interfaced to an 8-bit external device.

Code Example: /* example reading 256 bytes starting at 0x6400*/
unsigned char myByteArray[];
...
PMPMasterReadByteBlock(0x6400, 256, &myByteArray);
### PMPMasterReadWordBlock

**Description:** This function reads a block of 16-bit (word) data from an external device.

**Include:** plib.h

**Prototype:**

```c
void PMPMasterReadWordBlock(unsigned int address, unsigned int bytes, unsigned short* pDest);
```

**Arguments:**

- `address` External 16-bit starting address.
- `bytes` The number of words to read.
- `pDest` 16-bit (word) pointer to user memory where the data will be copied.

**Return Value:** None

**Remarks:** This function polls the PMP Busy flag to ensure any previous read or write operation has completed prior to reading a block of data. This function is intended for use when the PMP is interfaced to an 16-bit external device.

**Code Example:**

```c
/* example reading 16 words starting at 0x4000*/
unsigned char myWordArray[];
...
PMPMasterReadWordBlock(0x4000, 16, &myWordArray);
```

### PMPMasterWrite

**Description:** This function writes 8-,16-bit data to an external device.

**Include:** plib.h

**Prototype:**

```c
void PMPMasterWrite(unsigned int value);
```

**Arguments:**

- `value` An 8-,16 bit value to be written to an external device.

**Return Value:** None

**Remarks:** This function polls the PMP Busy flag to ensure any previous read or write operation has completed prior to writing to the PMDIN register. This function can be used when the PMP is interfaced to either 8-,16-bit external device.

**Code Example:**

```c
/* example using function in 16-bit PMP mode */
PMPMasterWrite(0x08FF);
```

```c
/* example using function in 8-bit PMP mode */
PMPMasterWrite(0x20)
```
**PMPMasterWriteByteBlock**

**Description:** This function writes a block of 8-bit (byte) data to an external device.

**Include:** plib.h

**Prototype:**

```c
void PMPMasterWriteByteBlock(unsigned int address, unsigned int bytes, unsigned char* pSrc);
```

**Arguments:**

- `address` External 16-bit starting address.
- `bytes` The number of bytes to write.
- `pSrc` 8-bit (byte) pointer to source data in user memory.

**Return Value:** None

**Remarks:** This function polls the PMP Busy flag to ensure any previous read or write operation has completed prior to reading a block of data. This function is intended for use when the PMP is interfaced to an 8-bit external device.

**Code Example:**

```c
/* example writing 64 bytes starting at 0x1000*/
unsigned char myByteArray[];
...
PMPMasterWriteByteBlock(0x1000, 64, &myByteArray);
```
### PMPMasterWriteWordBlock

**Description:** This function writes a block of 16-bit (word) data to an external device.

**Include:** plib.h

**Prototype:**

```c
void PMPMasterWriteWordBlock(unsigned int address, unsigned int words, unsigned char* pSrc);
```

**Arguments:**

- `address` External 16-bit starting address.
- `words` The number of words to write.
- `pSrc` 16-bit (word) pointer to source data in user memory.

**Return Value:** None

**Remarks:** This function polls the PMP Busy flag to ensure any previous read or write operation has completed prior to reading a block of data. This function is intended for use when the PMP is interfaced to an 16-bit external device.

**Code Example:**

```c
/* example writing 32 words starting at 0x8000*/
unsigned char myWordArray[];
...
PMPMasterWriteWordBlock(0x8000, 32, &myWordArray);
```

### mIsPMPBusy

**Description:** This macro provides the state of the PMP module busy flag

**Include:** plib.h

**Prototype:**

```c
void mIsPMPBusy(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** This macro provides PMMODE<BUSY> status bit.

**Notes:** The PMMODE.BUSY flag is only used in Master mode 1 and 2

**Code Example:**

```c
while(mIsPMPBusy());
```
### mPMPGetBusyFlag

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>This macro provides the state of the PMP module busy flag</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Include:</strong></td>
<td>plib.h</td>
</tr>
<tr>
<td><strong>Prototype:</strong></td>
<td>void mIsPMPBusy(void);</td>
</tr>
<tr>
<td><strong>Arguments:</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value:</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Remarks:</strong></td>
<td>Same macro functionality as &quot;mIsPMPBusy&quot;</td>
</tr>
<tr>
<td><strong>Notes</strong></td>
<td>The PMMODE.BUSY flag is only used in Master mode 1 and 2</td>
</tr>
<tr>
<td><strong>Code Example:</strong></td>
<td>while(mIsPMPBusy());</td>
</tr>
</tbody>
</table>
17.3 Slave Mode Functions and Macros

### mPMPSlaveRead

**Description:** This macro reads the slave input buffer.

**Include:** plib.h

**Prototype:**

```c
void mPMPSlaveRead(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** When operating in legacy slave mode, this macro provides the value in the PMPDIN register.

**Notes**

- This macro does not check the status of the PMSTAT.IBF (input buffer full) bit prior to reading the PMDIN register. It is recommended that the user verify PMSTAT<IBF> = 1 prior to reading the PMDIN register.
- If an external master write occurs before the current contents of the PMDIN register is performed, the IBOV flag will be set, indicating an overflow. This function does not check or modify the IBOV bit. Therefore the user should check for an overflow condition.

**Code Example:**

```c
/* example slave read */
unsigned char value;
...
value = mPMPSlaveRead();
```

### PMPSlaveReadBuffer

**Description:** This function reads one of four selected slave input buffers.

**Include:** plib.h

**Prototype:**

```c
unsigned int PMPSlaveReadBuffer(BUFFER buf)
```

**Arguments:**

- `buf` (enum) Slave buffer to read, 0..3.

**Return Value:** Value read from the selected input buffer.

**Remarks:** When operating in enhanced slave mode, this function reads the PMDIN input buffer register selected by the `buf` parameter and returns the 8-bit value.

**Code Example:**

```c
/* example reading slave buffer 3*/
unsigned char dataValue;
...
dataValue = PMPSlaveReadBufferN(3);
```
**PMPSlaveReadBuffers**

**Description:** This function reads all slave input buffers.

**Include:** plib.h

**Prototype:**

```c
unsigned int PMPSlaveReadBuffers(unsigned char* pDest)
```

**Arguments:**

- `pDest` : 8-bit (byte) pointer to user memory where the data will be copied.

**Return Value:** None

**Remarks:** When operating in buffered slave mode, this function reads all 4 slave data input buffers and copies to user memory specified by pointer.

**Code Example:**

```c
unsigned char dataOut[4];
...  
PMPSlaveReadBuffers(&dataOut);
```

---

**mIsPMPSlaveBufferFull**

**Description:** This macro provides the state of the slave Input Buffer Full status flag

**Include:** plib.h

**Prototype:**

```c
void mIsPMPSlaveBufferFull(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** This macro provides the PMSTAT<IBF> status bit

**Code Example:**

```c
while(!mIsPMPSlaveBufferFull());
```
### mPMPGetBufferFullFlags
**Description:** This macro provides the state of individual slave Input Buffer Full status flags

**Include:** plib.h

**Prototype:**
```c
void mPMPGetBufferFullFlags(BUFFER buf);
```

**Arguments:**
- `buf` (enum) The buffer register to write

**Remarks:** This macro provides PMSTAT<IBnF> status bits

**Notes**

**Code Example:**
```c
mPMPGetBufferFullFlags(BUF0);
```

### mIsPMPSlaveBufferOverflow
**Description:** This macro provides the state of the slave Input Buffer Overflow flag

**Include:** plib.h

**Prototype:**
```c
void mIsPMPSlaveBufferOverflow(void);
```

**Arguments:** None

**Remarks:** This macro provides PMSTAT<IBOV> status bit

**Notes**

**Code Example:**
```c
if(mIsPMPSlaveBufferOverflow());
```

### mPMPClearBufferOverflow
**Description:** This macro clears the slave Input Buffer Overflow flag

**Include:** plib.h

**Prototype:**
```c
void mPMPClearBufferOverflow(void);
```

**Arguments:** None

**Remarks:** This macro clears PMSTAT<IBF> status bit

**Notes**

**Code Example:**
```c
mPMPClearBufferOverflow();
```

### mPMPSlaveWrite
**Description:** This function writes to the slave output buffer.

**Include:** plib.h

**Prototype:**
```c
void mPMPSlaveWrite(unsigned char value);
```
**mPMPSlaveWrite** (Continued)

**Arguments:**  
value  
8-bit value to load into the slave output buffer

**Return Value:**  
None

**Remarks:**  
When operating in legacy slave mode, this function writes the data value to the PMDOUT buffer register.

**Notes**  
This function does not check the status of the PMSTAT.OBE (output buffer empty) bit prior to writing to the PMDOUT register. Therefore the user should check PMSTAT<OBIE> bit = 1 prior to writing the PMDOUT register.

**Code Example:**  
/* example slave write */  
mPMPSlaveWrite(0xFF);  

**PMPSlaveWriteBuffer**

**Description:**  
This function writes one of four selected slave output buffers.

**Include:**  
plib.h

**Prototype:**  
void PMPSlaveWriteBuffer(BUFFER buf, unsigned int value)

**Arguments:**  
buf  
The buffer register to write  
value  
The 8-bit (byte) value to write

**Return Value:**  
None

**Remarks:**  
When operating in enhanced slave mode, this function writes a byte to the PMDOUT output buffer register selected by the buf parameter.

**Code Example:**  
/* example load 0x55 in slave buffer 0 */  
PMPSlaveWriteBuffer(0,0x55);
**PMPSlaveWriteBuffers**

**Description:** This function writes to all 4 slave output buffers.

**Include:** plib.h

**Prototype:**

```c
void PMPSlaveWriteBuffers(unsigned char* pSrc)
```

**Arguments:**

- `pSrc` : 8-bit (byte) pointer to source data in user memory.

**Return Value:** None

**Remarks:** When operating in enhanced slave mode, this function writes 4 bytes, pointed to by `pSrc` parameter, to PMDOUT output buffers.

**Code Example:**

```c
/* example load 4 values in slave output buffers */
unsigned char dataOut[4];
...
PMPSlaveWriteBuffers(&dataOut);
```

---

**mIsPMPSlaveBufferEmpty**

**Description:** This macro provides the state of the slave Output Buffer Empty status flag.

**Include:** plib.h

**Prototype:**

```c
void mIsPMPSlaveBufferEmpty(void);
```

**Arguments:** None

**Remarks:** This macro provides PMSTAT<OBF> status bit.

**Notes**

**Code Example:**

```c
if(mIsPMPSlaveBufferEmpty());
....
else
....
```
### mPMPGetBufferEmptyFlags

**Description:** This macro provides the state of individual slave Output Buffer Empty status flags

**Include:** plib.h

**Prototype:**
```c
void mPMPGetBufferEmptyFlags(void);
```

**Arguments:** None

**Remarks:** This macro provides PMSTAT<OBnE> status bits

**Notes**

**Code Example:**
```c
mPMPGetBufferEmptyFlags(BUF3);
```

### mIsPMPSlaveBufferUnderflow

**Description:** This macro provides the state of the slave Output Buffer Underflow status flag

**Include:** plib.h

**Prototype:**
```c
void mIsPMPSlaveBufferUnderflow(void);
```

**Arguments:** None

**Remarks:** This macro provides PMSTAT.OBUF status bit

**Notes**

**Code Example:**
```c
if(mIsPMPSlaveBufferUnderflow());
```

### mPMPClearBufferUnderflow

**Description:** This macro clears the slave Output Buffer Underflow flag.

**Include:** plib.h

**Prototype:**
```c
void mPMPClearBufferUnderflow(void);
```

**Arguments:** None

**Remarks:** This macro clears PMSTAT<OBUF> status bit

**Notes**

**Code Example:**
```c
PMPClearBufferUnderflow();
```
17.4 Example of Use

// Example 1 demonstrates legacy slave mode configuration and use.
#include <plib.h>

/* select legacy slave mode, "active lo" logic, with interrupts */
#define PMP_CONTROL   PMP_READ_POL_LO | PMP_WRITE_POL_LO |\
                     PMP_CS1_POL_LO
#define PMP_MODE      PMP_MODE_SLAVE
#define PMP_PORT      0x0000
#define PMP_ADDR      0x0000
#define PMP_INT       PMP_INT_PRI_3 | PMP_INT_ON

unsigned char data;

int main(void)
{
    mPMPOpen(PMP_CONTROL, PMP_MODE, PMP_PORT, PMP_INT);
    // poll for external master device to write something...
    while(!mIsPMPSlaveBufferFull());
    data = mPMPSlaveRead();
    ...
    // later, prepare data for the external master to read
    // be sure to check if output buffer is empty before writing
    while(!mIsPMPSlaveBufferEmpty());
    mPMPSlaveWrite(0x22);
    ...
}
19.1 RTCC FUNCTIONS

This document provides a list and a description of the interface functions that are part of the RTCC API Peripheral Library. It is intended as a quick reference to the user of the RTCC API. So, it is a complete specification of all the functions provided as well as being a guide to using these functions.

19.1.1 High Level Control Functions

The following set of functions control the initialization and shutdown operation of the RTCC.

### RtccInit

| Description | The function initializes the RTCC device. It starts the RTCC clock, enables the RTCC and disables RTCC write. Disables the Alarm and the OE. Clears the alarm interrupt flag. |
| Include | plib.h |
| Prototype | rtccRes RtccInit(void); |
| Arguments | None |
| Return Value | - RTCC_CLK_ON if the RTCC clock is actually running |
| Remarks | This function has to be called before using RTCC module services. It usually takes 4x256 clock cycles (approx 31.2 ms) for the oscillator signal to be available to the RTCC. The user must make sure that the clock is actually running using RtccGetClkStat() before expecting the RTCC to count. |
| Source File | rtcc_init_lib.c |
| Coding Example | rtccRes res=RtccInit(); if(res==RTCC_CLK_ON) { // RTCC clock is running ...} |

### RtccOpen

| Description | The function initializes the RTCC device. It starts the RTCC clock, sets the desired time and calibration and enables the RTCC. Disables the Alarm and the OE and further RTCC writes. Clears the alarm interrupt flag. |
| Include | plib.h |
| Prototype | rtccRes RtccOpen((unsigned long tm, unsigned long dt, int drift); |
| Arguments | tm - an unsigned long containing the fields of a valid rtccTime structure: |
| | - sec: BCD codification, 00-59 |
| | - min: BCD codification, 00-59 |
| | - hour: BCD codification, 00-24 |
| dt - the date value to be set containing the valid fields of a rtccDate structure: |
| | - wday: BCD codification, 00-06 |
| | - mday: BCD codification, 01-31 |
| | - mon: BCD codification, 01-12 |
| | - year: BCD codification, 00-99 |
| drift - value to be added/subtracted to perform calibration. The drift value acts as a signed value, [-512, +511], 0 not having any effect. |

### 19.1.2 Time and Alarm Functions

These functions deal with the setting and retrieving of the RTCC current time and alarm time.

#### RtccSetTime

This function sets the current time in the RTCC device.

**Include:** `plib.h`

**Prototype:**

```c
void RtccSetTime(unsigned long tm);
```

**Arguments:**

- `tm` - an unsigned long containing the fields of a valid `rtccTime` structure:
  - `sec`: BCD codification, 00-59
  - `min`: BCD codification, 00-59
  - `hour`: BCD codification, 00-24

**Return Value:**

None

---

#### RtccShutdown

The function shuts down the RTCC device. It stops the RTCC clock, sets the RTCC Off and disables RTCC write. Disables the Alarm and the OE. Clears the alarm interrupt flag.

**Include:** `plib.h`

**Prototype:**

```c
void RtccShutdown(void);
```

**Arguments:** None

**Return Value:** None

**Remarks:** After using this function `RtccInit()` has to be called again to be able to use the RTCC module services.

**Source File:** `rtcc_shutdown_lib.c`

**Coding Example:**

```c
RtccShutdown();
```

---

#### Coding Example

```c
t RCCDate dt; dt.wday=05; dt.mday=0x28; dt.mon=0x2; dt.year=0; rtccTime tm; tm.sec=0x15;
  tm.min=0x30; tm.hour=01; rtccRes res=RtccOpen(tm.l, dt.l, 10);
  or
  rtccRes res=RtccOpen(0x01301500, 0x00022805, 10);
```
### Remarks:
- The write is successful only if Wr Enable is set. The function will enable the write itself, if needed.
- The device could be stopped in order to safely perform the update of the RTC time register. However, the device status will be restored but the routine won't wait for the CLK to be running before returning. User has to check RtccGetClkStat() (will take approx 30us).
- The routine could disable the interrupts for a very short time to be able to update the time and date registers.

**Source File:** rtcc_set_time_lib.c

**Coding Example:**
```c
rtccTime tm; tm.sec=0x15; tm.min=0x30; tm.hour=01; RtccSetTime(tm.l);
or
RtccSetTime(0x01301500);
```

---

### RtccGetTime

**Description:**
The function returns the current time of the RTCC device.

**Include:**
plib.h

**Prototype:**
```c
unsigned long RtccGetTime(void);
```

**Arguments:** None

**Return Value:**
The current value of the time which can be safely casted to an rtccTime structure.

**Remarks:**
- The function makes sure that the read value is valid. It avoids waiting for the RTCSYNC to be clear by performing successive reads.

**Source File:** rtcc.h

**Coding Example:**
```c
rtccTime tm; tm.l=RtccGetTime();
```

---

### RtccSetDate

**Description:**
The function sets the current date in the RTCC device.

**Include:**
plib.h

**Prototype:**
```c
void RtccSetDate(unsigned long dt);
```

**Arguments:**
- dt - the date value to be set containing the valid fields of a rtccDate structure:
  - wday: BCD codification, 00-06
  - mday: BCD codification, 01-31
  - mon: BCD codification, 01-12
  - year: BCD codification, 00-99

**Return Value:** None

**Remarks:**
- The write is successful only if Wr Enable is set. The function will enable the write itself, if needed.
- The device could be stopped in order to safely perform the update of the RTC time register. However, the device status will be restored but the routine won't wait for the CLK to be running before returning. User has to check RtccGetClkStat() (will take approx 30us).
- The routine could disable the interrupts for a very short time to be able to update the time and date registers.

**Source File:** rtcc_set_date_lib.c
### RtccGetDate

**Description:** The function returns the current date of the RTCC device. Can be safely cast into rtccDate.

**Include:** plib.h

**Prototype:**

```c
unsigned long RtccGetDate(void);
```

**Arguments:** None

**Return Value:** an unsigned long representing the current date:

- `wday`: BCD codification, 00-06
- `mday`: BCD codification, 01-31
- `mon`: BCD codification, 01-12
- `year`: BCD codification, 00-99

**Remarks:** The function makes sure that the read value is valid. It avoids waiting for the RTCSYNC to be clear by performing successive reads.

**Source File:** rtcc.h

**Coding Example:**

```c
rtccDate dt; dt.l = RtccGetDate();
```

---

### RtccSetTimeDate

**Description:** The function sets the current time and date in the RTCC device.

**Include:** plib.h

**Prototype:**

```c
void RtccSetTimeDate(unsigned long tm, unsigned long dt);
```

**Arguments:**

- `tm` - the time value to be set, a valid rtccTime structure having proper values:
  - `sec`: BCD codification, 00-59
  - `min`: BCD codification, 00-59
  - `hour`: BCD codification, 00-24

- `dt` - the date value to be set, a valid rtccDate structure having proper values:
  - `wday`: BCD codification, 00-06
  - `mday`: BCD codification, 01-31
  - `mon`: BCD codification, 01-12
  - `year`: BCD codification, 00-99

**Return Value:** None

**Remarks:**

- The write is successful only if Wr Enable is set. The function will enable the write itself, if needed.
- The device could be stopped in order to safely perform the update of the RTC time register. However, the device status will be restored but the routine won’t wait for the CLK to be running before returning. User has to check RtccGetClkStat() (will take approx 30us).
- The routine could disable the interrupts for a very short time to be able to update the time and date registers.
RtccGetTimeDate

Description: The function updates the user supplied union/structures with the current time and date of the RTCC device.

Include: plib.h

Prototype: void RtccGetTimeDate(rtccTime* pTm, rtccDate* pDt);

Arguments:
- pTm - pointer to a rtccTime union to store the current time:
  - sec: BCD codification, 00-59
  - min: BCD codification, 00-59
  - hour: BCD codification, 00-24
- pDt - pointer to a rtccDate union to store the current date:
  - wday: BCD codification, 00-06
  - mday: BCD codification, 01-31
  - mon: BCD codification, 01-12
  - year: BCD codification, 00-99

Return Value: None

Remarks: - The function makes sure that the read value is valid. It avoids waiting for the RTCSYNC to be clear by performing successive reads.

Source File: rtcc.h

Coding Example: rtccTime tm; rtccDate dt; RtccGetTimeDate(&tm, &dt);

RtccSetAlarmTime

Description: The function sets the current alarm time in the RTCC device.

Include: plib.h

Prototype: void RtccSetAlarmTime(unsigned long tm);

Arguments: tm - the alarm time to be set, a valid rtccTime structure having proper values:
- sec: BCD codification, 00-59
- min: BCD codification, 00-59
- hour: BCD codification, 00-24

Return Value: None

Remarks: - The function might wait for the proper Alarm window to safely perform the update of the ALRMTIME register.
- Interrupts are disabled shortly when properly probing the RTCSYNC/ALRMSYNC needed.

Source File: rtcc_set_alarm_time_lib.c
mRtccGetAlarmTime

Description: The macro returns the current alarm time of the RTCC device.

Include: plib.h

Prototype: unsigned long mRtccGetAlarmTime(void);

Arguments: None

Return Value: the current alarm time, a value that can be safely cast into a rtccTime union:
- sec: BCD codification, 00-59
- min: BCD codification, 00-59
- hour: BCD codification, 00-24

Remarks: None

Source File: rtcc.h

Coding Example:

rtccTime tm; tm.l=mRtccGetAlarmTime();

RtccSetAlarmDate

Description: The function sets the alarm date in the RTCC device.

Include: plib.h

Prototype: void RtccSetAlarmDate(unsigned long dt);

Arguments: dt - value of the alarm date, a valid rtccDate formatted structure having proper values:
- wday: BCD codification, 00-06
- mday: BCD codification, 01-31
- mon: BCD codification, 01-12

Return Value: None

Remarks:
- The function might wait for the proper Alarm window to safely perform the update of the ALRMDATE register.
- Interrupts are disabled shortly when properly probing the RTCSYNC/ALRMSYNC needed.
- Note that the alarm date does not contain a year field.

Source File: rtcc_set_alarm_date_lib.c

Coding Example:

rtccDate dt; dt.wday=0; dt.mday=0x12; dt.mon=0x12; RtccSetAlarmDate(dt.l);

or
RtccSetAlarmDate(0x121200);
### mRtccGetAlarmDate

**Description:**
The macro returns the current alarm date of the RTCC device.

**Include:**
plib.h

**Prototype:**
unsigned long mRtccGetAlarmDate(void);

**Arguments:**
None

**Return Value:**
The current alarm date. Can be safely cast into an rtccDate:
- wday: BCD codification, 00-06
- mday: BCD codification, 01-31
- mon: BCD codification, 01-12

**Remarks:**
None

**Source File:**
rtcc.h

**Coding Example:**
rtccDate dt; dt.l=mRtccGetAlarmDate();

---

### RtccSetAlarmTimeDate

**Description:**
The function sets the current alarm time and date in the RTCC device.

**Include:**
plib.h

**Prototype:**
void RtccSetAlarmTimeDate(unsigned long tm, unsigned long dt);

**Arguments:**
- tm - the alarm time to be set, a valid rtccTime structure having proper values:
  - sec: BCD codification, 00-59
  - min: BCD codification, 00-59
  - hour: BCD codification, 00-24
- dt - the alarm date to be set, a valid rtccDate structure having proper values:
  - wday: BCD codification, 00-06
  - mday: BCD codification, 01-31
  - mon: BCD codification, 01-12

**Return Value:**
None

**Remarks:**
- The function might wait for the proper Alarm window to safely perform the update of the ALRMTIME, ALRMDATE registers.
- Interrupts are disabled shortly when properly probing the RTCSYNC/ALRMSYNC needed.
- Note that the alarm time does not contain a year field.

**Source File:**
rtcc_set_alarm_time_date_lib.c

**Coding Example:**
rtccTime tm; tm.sec=0; tm.min=0x59; tm.hour=0x23; rtccDate dt; dt.wday=0; dt.mday=0x12; dt.mon=0x12; RtccSetAlarmTimeDate(tm.l, dt.l);
or
RtccSetAlarmTimeDate(0x235900, 0x121200);
RtccGetAlarmTimeDate

Description: The function updates the user supplied union/structures with the current alarm time and date of the RTCC device.

Include:

plib.h

Prototype:

void RtccGetAlarmTimeDate(rtccTime* pTm, rtccDate* pDt);

Arguments:

pTm - pointer to a rtccTime union to store the alarm time:
- sec: BCD codification, 00-59
- min: BCD codification, 00-59
- hour: BCD codification, 00-24
pDt - pointer to a rtccDate union to store the alarm date:
- wday: BCD codification, 00-06
- mday: BCD codification, 01-31
- mon: BCD codification, 01-12

Return Value: None

Remarks: None

Source File: rtcc.h

Coding Example:

rtccTime tm; rtccDate dt; RtccGetAlarmTimeDate(&tm, &dt);

RtccWeekDay

Description: The function calculates the week of the day for new style dates, beginning at 14 Sep 1752.

Based on an algorithm by Lewis Carroll.

Include:

plib.h

Prototype:

int RtccWeekDay(int year, int month, int day);

Arguments:

year - year value
month - month value, 1-12
day - day value, 1-31

Return Value: the week of the day, 0 for Sun, 1 for Mon and so on

Remarks: None

Source File: rtcc_weekday_lib.c

Coding Example:

int weekDay=RtccWeekDay(2004, 02, 28);

19.1.3 Alarm Control and status functions

The following set of functions control the operation of the RTCC Alarm. They also return the current status of the RTCC alarm settings.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Include:</th>
<th>Prototype:</th>
<th>Arguments:</th>
<th>Return Value:</th>
<th>Remarks:</th>
<th>Source File:</th>
<th>Coding Example:</th>
</tr>
</thead>
</table>
| RtccAlarmEnable   | The function enables the alarm of the RTCC device.                          | plib.h     | void RtccAlarmEnable(void);                  | None       | None         | - The function might wait for the proper Alarm window to safely perform the update of the RTCALRM register.  
- Interrupts are disabled shortly when properly probing the RTCSYNC/ALRMSYNC needed. | rtcc_alarm_enable_lib.c       | RtccAlarmEnable(); |
| RtccAlarmDisable  | The function disables the alarm of the RTCC device.                         | plib.h     | void RtccAlarmDisable(void);                 | None       | None         | - The function might wait for the proper Alarm window to safely perform the update of the RTCALRM register.  
- Interrupts are disabled shortly when properly probing the RTCSYNC/ALRMSYNC needed. | rtcc_alarm_disable_lib.c      | RtccAlarmDisable(); |
| RtccGetAlarmEnable| The function returns the current alarm status of the RTCC device.           | plib.h     | int RtccGetAlarmEnable(void);               | None       | true- if alarm is enabled  
false- if alarm is disabled | None                                                                                     | rtcc.h                      | int isAlrmEnabled=RtccGetAlarmEnable(); |
### RtccChimeEnable

**Description:** The function enables the chime alarm of the RTCC device.

**Include:**
- plib.h

**Prototype:**
- void RtccSetChimeEnable(bool enable, bool dsblAlrm);

**Arguments:**
- None

**Return Value:**
- None

**Remarks:**
- The function might wait for the proper Alarm window to safely perform the update of the RTCALRM register.
- Interrupts are disabled shortly when properly probing the RTCSYNC/ALRMSYNC needed.

**Source File:**
- rtcc_chime_enable_lib.c

**Coding Example:**
- RtccChimeEnable();

### RtccChimeDisable

**Description:** The function disables the chime alarm of the RTCC device.

**Include:**
- plib.h

**Prototype:**
- void RtccSetChimeEnable(bool enable, bool dsblAlrm);

**Arguments:**
- None

**Return Value:**
- None

**Remarks:**
- The function might wait for the proper Alarm window to safely perform the update of the RTCALRM register.
- Interrupts are disabled shortly when properly probing the RTCSYNC/ALRMSYNC needed.

**Source File:**
- rtcc_chime_disable_lib.c

**Coding Example:**
- RtccChimeDisable();

### RtccGetChimeEnable

**Description:** The function returns the chime alarm of the RTCC device.

**Include:**
- plib.h

**Prototype:**
- int RtccGetChimeEnable(void);

**Arguments:**
- None

**Return Value:**
- true- if chime is enabled
  - false- if chime is disabled
RtccSetAlarmRpt

Description:  The function sets the RTCC alarm repeat rate.
Include:    plib.h
Prototype:    void RtccSetAlarmRpt(rtccRepeat rpt);
Arguments:    rpt - value of the desired alarm repeat rate:
  RTCC_RPT_HALF_SEC - repeat alarm every half second
  RTCC_RPT_SEC - repeat alarm every second
  RTCC_RPT_TEN_SEC - repeat alarm every ten seconds
  RTCC_RPT_MIN - repeat alarm every minute
  RTCC_RPT_TEN_MIN - repeat alarm every ten minutes
  RTCC_RPT_HOUR - repeat alarm every hour
  RTCC_RPT_DAY - repeat alarm every day
  RTCC_RPT_WEEK - repeat alarm every week
  RTCC_RPT_MON - repeat alarm every month
  RTCC_RPT_YEAR - repeat alarm every year
Return Value:    None
Remarks:    - The function might wait for the proper Alarm window to safely perform the update of the
  RTCALRM register.
    - Interrupts are disabled shortly when properly probing the RTCSYNC/ALRMSYNC
      needed.
Source File:    rtcc_set_alarm_rpt_lib.c
Coding Example:    RtccSetAlarmRpt(RTCC_RPT_MIN);

RtccGetAlarmRpt

Description:  The function returns the current RTCC alarm repeat rate.
Include:    plib.h
Prototype:    rtccRepeat RtccGetAlarmRpt(void);
Arguments:    None
Return Value: The value of the current alarm repeat rate:

- RTCC_RPT_HALF_SEC – alarm is repeated every half second
- RTCC_RPT_SEC – alarm is repeated every second
- RTCC_RPT_TEN_SEC - alarm is repeated every ten seconds
- RTCC_RPT_MIN - alarm is repeated every minute
- RTCC_RPT_TEN_MIN - alarm is repeated every ten minutes
- RTCC_RPT_HOUR - alarm is repeated every hour
- RTCC_RPT_DAY - alarm is repeated every day
- RTCC_RPT_WEEK - alarm is repeated every week
- RTCC_RPT_MON - alarm is repeated every month
- RTCC_RPT_YEAR - alarm is repeated every year

Remarks: None

Source File: rtcc.h

Coding Example: rtccRepeat rptAlrm=RtccGetAlarmRpt();

---

RtccSetAlarmRptCount

Description: The function sets the RTCC alarm repeat count.

Include: plib.h

Prototype: void RtccSetAlarmRptCount(int rptCnt);

Arguments: rpt - value of the desired alarm repeat count, less than 256

The number of alarm triggers will be rptCnt+1:
- one alarm trigger if rptCnt==0
- ....
- 256 alarm triggers if rptCnt=255

Return Value: None

Remarks:
- rptCnt will be truncated to fit into 8 bit representation.
- The function might wait for the proper Alarm window to safely perform the update of the RTCALRM register.
- Interrupts are disabled shortly when properly probing the RTCSYNC/ALRMSYNC needed.
- If rptCnt is 0, there will be one alarm trigger.

Source File: rtcc_set_alarm_rpt_count_lib.c

Coding Example: RtccSetAlarmRptCount(10);

---

RtccGetAlarmRptCount

Description: The function reads the RTCC alarm repeat counter.

Include: plib.h

Prototype: int RtccGetAlarmRptCount(void);

Arguments: None

Return Value: The current alarm repeat count
19.1.4 Low Level Control and Status Function

The following set of functions provides a low level interface for controlling the operation of the RTCC. They also return the current status of certain RTCC settings as well as the status of internal RTCC bits.

Remarks:
The reading is affected by the status of RTCALRM.ALRMSYNC bit. Double readings are performed.

Source File:
rtcc.h

Coding Example:
int alrmRptCnt=RtccGetAlarmRptCount();

### RtccEnable

<table>
<thead>
<tr>
<th>Description:</th>
<th>The function enables the RTCC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>plib.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td>rtccRes RtccEnable(void)</td>
</tr>
<tr>
<td>Arguments:</td>
<td>None</td>
</tr>
<tr>
<td>Return Value:</td>
<td>- RTCC_CLK_ON if the RTCC clock is actually running</td>
</tr>
<tr>
<td></td>
<td>- RTCC_SOSC_NRDY if the SOSC is not running</td>
</tr>
<tr>
<td></td>
<td>- RTCC_CLK_NRDY if the RTCC clock is not running</td>
</tr>
<tr>
<td></td>
<td>- RTCC_WR_DSBL if the write is disabled</td>
</tr>
<tr>
<td>Remarks:</td>
<td>- The write operations have to be enabled in order to be able to toggle the ON control bit. Otherwise the function will fail. See RtccWrEnable() function.</td>
</tr>
<tr>
<td></td>
<td>- The function doesn't wait for the RTC clock to be on.</td>
</tr>
<tr>
<td>Source File:</td>
<td>rtcc_enable_lib.c</td>
</tr>
<tr>
<td>Coding Example:</td>
<td>rtccRes clkStat=RtccEnable();</td>
</tr>
</tbody>
</table>

### RtccDisable

<table>
<thead>
<tr>
<th>Description:</th>
<th>The function disables the RTCC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>plib.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td>int RtccDisable(void)</td>
</tr>
<tr>
<td>Arguments:</td>
<td>None</td>
</tr>
<tr>
<td>Return Value:</td>
<td>TRUE if the RTCC was disabled,</td>
</tr>
<tr>
<td></td>
<td>FALSE if the write is disabled.</td>
</tr>
</tbody>
</table>
- The write operations have to be enabled in order to be able to toggle the ON control bit. Otherwise the function will fail. See RtccWrEnable() function.
- When ON control bit is set to 0, RTCCON.RTCSYNC, RTCCON.HALFSEC and RTCCON.RTCOE are asynchronously reset.
- The function waits for the RTC clock to be off.

Source File: rtcc_disable_lib.c
Coding Example: int isDisabled=RtccDisable();

mRtccGetEnable
Description: The macro returns the enabled/disabled status of the RTCC module (i.e. the RTCCON.ON bit anded with RTCCLKON).
Include: plib.h
Prototype: int mRtccGetEnable(void)
Arguments: None
Return Value: true- if RTCC is enabled false- otherwise
Remarks: None
Source File: rtcc.h
Coding Example: int isEnabled=mRtccGetEnable();

RtccGetClkStat
Description: The function returns the status of the RTCC clock (the RTCCON.ON bit anded with RTCCLKON).
Include: plib.h
Prototype: rtccRes RtccGetClkStat(void);
Arguments: None
Return Value: - RTCC_CLK_ON if the RTCC clock is actually running
             - RTCC_SOSC_NRDY if the SOSC is not running
             - RTCC_CLK_NRDY if the RTCC clock is not running
Remarks: None
Source File: rtcc.h
Coding Example: rtccRes clkStat=RtccGetClkStat(); if(clkStat==RTCC_CLK_ON) {// clock ok...}
**RtccSetCalibration**

**Description:** The function updates the value that the RTCC uses in the auto-adjust feature, once every minute. The drift value acts as a signed value, [-512, +511], 0 not having any effect.

**Include:** `plib.h`

**Prototype:** `void RtccSetCalibration(int drift);`

**Arguments:** `drift` - value to be added/subtracted to perform calibration. The drift value acts as a signed value, [-512, +511], 0 not having any effect.

**Return Value:** None

**Remarks:**
- Writes to the RTCCON.CAL[9:0] register should only occur when the timer is turned off or immediately or after the edge of the seconds pulse (except when SECONDS=00 - due to the possibility of the auto-adjust event). In order to speed-up the process, the API function performs the reading of the HALFSEC field.
- The function may block for half a second, worst case, when called at the start of the minute.
- A write to the SECONDS value resets the state of the calibration and the prescaler. If calibration just occurred, it will occur again at the prescaler rollover.
- Interrupts can not be disabled for such a long period. However, long interrupt routines can interfere with the proper functioning of the device. Care must be taken.

**Source File:** `rtcc_set_calibration_lib.c`

**Coding Example:** `RtccSetCalibration (200);`

---

**mRtccGetCalibration**

**Description:** The macro returns the value that the RTCC uses in the auto-adjust feature, once every minute. The calibration value is a signed 10 bits value, [-512, +511].

**Include:** `plib.h`

**Prototype:** `int mRtccGetCalibration(void);`

**Arguments:** None

**Return Value:** Current value of the RTCC calibration field.

**Remarks:** None

**Source File:** `rtcc.h`

**Coding Example:** `int currCal=mRtccGetCalibration();`

---

**RtccWrEnable**

**Description:** The function enables the updates to the RTCC time registers and ON control bit.

**Include:** `plib.h`

**Prototype:** `void RtccWrEnable();`

**Arguments:** None

**Return Value:** None
- The write can be enabled by performing a specific unlock sequence. In order to succeed, this sequence need not be interrupted by other memory accesses (DMA transfers, interrupts, etc).
- Interrupts and DMA transfers that might disrupt the write unlock sequence are disabled shortly for properly unlocking the device.

**Source File:** rtcc_wr_enable_lib.c

**Coding Example:**
```
RtccWrEnable ();
```

## mRtccWrDisable

**Description:** The macro performs the system lock sequence so that further updates to the RTCC time registers and ON control bit are disabled.

**Include:** plib.h

**Prototype:** void mRtccWrDisable();

**Arguments:** None

**Return Value:** None

**Remarks:** None

**Source File:** rtcc.h

**Coding Example:**
```
mRtccWrDisable ();
```

## mRtccGetWrEnable

**Description:** The macro returns the current status of the RTCC write enable bit.

**Include:** plib.h

**Prototype:** int mRtccGetWrEnable(void);

**Arguments:** None

**Return Value:**
- true- if RTCC write is enabled
- false- otherwise

**Remarks:** None

**Source File:** rtcc.h

**Coding Example:**
```
int isWrEnabled=mRtccGetWrEnable();
```

## mRtccGetSync

**Description:** The macro returns the current status of the RTCC Sync bit.

**Include:** plib.h
**Prototype:**

```c
int nRtccGetSync(void);
```

**Arguments:**
None

**Return Value:**
true- if RTCC Sync is asserted
false- otherwise

**Remarks:**
None

**Source File:**
rtcc.h

**Coding Example:**
```c
int isSync=mRtccGetSync();
```

---

**mRtccGetHalfSecond**

**Description:**
The macro returns the current status of the RTCC HalfSec bit.

**Include:**
plib.h

**Prototype:**
```c
int mRtccGetHalfSecond(void);
```

**Arguments:**
None

**Return Value:**
true- if RTCC HalfSec is asserted
false- otherwise

**Remarks:**
None

**Source File:**
rtcc.h

**Coding Example:**
```c
int is2HalfSec=mRtccGetHalfSecond();
```

---

**mRtccGetAlrmSync**

**Description:**
The macro returns the current status of the RTCALRM ALRMSYNC bit.

**Include:**
plib.h

**Prototype:**
```c
int mRtccGetAlrmSync(void);
```

**Arguments:**
None

**Return Value:**
true- if RTCC AlrmSync is asserted
false- otherwise

**Remarks:**
None

**Source File:**
rtcc.h

**Coding Example:**
```c
int isAlrmSync=mRtccGetAlrmSync();
```

---

**mRtccSelectSecPulseOutput**

**Description:**
The macro selects the seconds clock pulse as the function of the RTCC output pin.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Include</th>
<th>Prototype</th>
<th>Arguments</th>
<th>Return Value</th>
<th>Remarks</th>
<th>Source File</th>
<th>Coding Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>mRtccSelectSecPulseOutput()</td>
<td>The macro selects the alarm pulse as the function of the RTCC output pin.</td>
<td>plib.h</td>
<td>void mRtccSelectSecPulseOutput(void);</td>
<td>None</td>
<td>None</td>
<td>The RTCC has to be enabled for the output to actually be active.</td>
<td>rtcc.h</td>
<td>mRtccSelectSecPulseOutput();</td>
</tr>
<tr>
<td>mRtccSelectAlarmPulseOutput()</td>
<td>The macro selects the alarm pulse as the function of the RTCC output pin.</td>
<td>plib.h</td>
<td>void mRtccSelectAlarmPulseOutput(void);</td>
<td>None</td>
<td>None</td>
<td>The RTCC has to be enabled for the output to actually be active.</td>
<td>rtcc.h</td>
<td>mRtccSelectAlarmPulseOutput();</td>
</tr>
<tr>
<td>RtccAlarmPulseHigh()</td>
<td>The function sets the initial value of the output Alarm Pulse to logic 1.</td>
<td>plib.h</td>
<td>void RtccAlarmPulseHigh(void);</td>
<td>None</td>
<td>None</td>
<td>- The RTCC has to be enabled for the output to actually be active.</td>
<td>rtcc_alarm_pulse_high_lib.c</td>
<td>RtccAlarmPulseHigh();</td>
</tr>
<tr>
<td>RtccAlarmPulseLow()</td>
<td>The function sets the initial value of the output Alarm Pulse to logic 0.</td>
<td>plib.h</td>
<td></td>
<td>None</td>
<td>None</td>
<td>- The alarm has to be disabled to be able to change the status of the Alarm Pulse</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototype:</td>
<td>void RtccAlarmPulseLow(void);</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>---------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments:</td>
<td>None</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Value:</td>
<td>None</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remarks:</td>
<td>- The RTCC has to be enabled for the output to actually be active.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- The alarm has to be disabled to be able to change the status of the Alarm Pulse</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source File:</td>
<td>rtcc_alarm_pulse_low_lib.c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coding Example:</td>
<td>RtccAlarmPulseLow ();</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**RtccAlarmPulseToggle**

<table>
<thead>
<tr>
<th>Description:</th>
<th>The function toggles the value of the output Alarm Pulse.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>plib.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td>void RtccAlarmPulseToggle(void);</td>
</tr>
<tr>
<td>Arguments:</td>
<td>None</td>
</tr>
<tr>
<td>Return Value:</td>
<td>None</td>
</tr>
<tr>
<td>Remarks:</td>
<td>- The RTCC has to be enabled for the output to actually be active.</td>
</tr>
<tr>
<td></td>
<td>- The alarm has to be disabled to be able to change the status of the Alarm Pulse</td>
</tr>
<tr>
<td>Source File:</td>
<td>rtcc_alarm_pulse_toggle_lib.c</td>
</tr>
<tr>
<td>Coding Example:</td>
<td>RtccAlarmPulseToggle ();</td>
</tr>
</tbody>
</table>

**mRtccGetAlarmPulse**

<table>
<thead>
<tr>
<th>Description:</th>
<th>The macro returns the current state of the output Alarm Pulse.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>plib.h</td>
</tr>
<tr>
<td>Prototype:</td>
<td>int mRtccGetAlarmPulse(void);</td>
</tr>
<tr>
<td>Arguments:</td>
<td>None</td>
</tr>
<tr>
<td>Return Value:</td>
<td>None</td>
</tr>
<tr>
<td>Remarks:</td>
<td>None</td>
</tr>
<tr>
<td>Source File:</td>
<td>rtcc.h</td>
</tr>
<tr>
<td>Coding Example:</td>
<td>int alrmPulse=mRtccGetAlarmPulse();</td>
</tr>
</tbody>
</table>

**mRtccOutputEnable**

<table>
<thead>
<tr>
<th>Description:</th>
<th>The macro enables the Output pin of the RTCC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>plib.h</td>
</tr>
</tbody>
</table>
### mRtccOutputEnable

**Prototype:**

```c
void mRtccOutputEnable(void);
```

**Arguments:**

None

**Return Value:**

None

**Remarks:**

The RTCC has to be enabled for the output to actually be active.

**Source File:**

rtcc.h

**Coding Example:**

```c
mRtccOutputEnable();
```

### mRtccOutputDisable

**Description:**

The macro disables the Output pin of the RTCC.

**Include:**

plib.h

**Prototype:**

```c
void mRtccOutputDisable (void);
```

**Arguments:**

None

**Return Value:**

None

**Remarks:**

None

**Source File:**

rtcc.h

**Coding Example:**

```c
mRtccOutputDisable();
```

### mRtccGetOutputEnable

**Description:**

The macro returns the enabled/disabled status of the RTCC Output pin.

**Include:**

plib.h

**Prototype:**

```c
int mRtccGetOutputEnable (void);
```

**Arguments:**

None

**Return Value:**

true if Output is enabled, false otherwise.

**Remarks:**

None

**Source File:**

rtcc.h

**Coding Example:**

```c
int isOutEnabled=mRtccGetOutputEnable();
```

### 19.1.5 Interrupt related functions

### mRtccGetIntFlag

**Description:**

This macro reads the interrupt controller to check if the RTCC interrupt flag is set.
<table>
<thead>
<tr>
<th><strong>mRtccGetIntFlag</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Include:</strong></td>
</tr>
<tr>
<td><strong>Prototype:</strong></td>
</tr>
<tr>
<td><strong>Arguments:</strong></td>
</tr>
<tr>
<td><strong>Return Value:</strong></td>
</tr>
<tr>
<td><strong>Remarks:</strong></td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
</tr>
<tr>
<td><strong>Coding Example:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>mRtccEnableInt</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Include:</strong></td>
</tr>
<tr>
<td><strong>Prototype:</strong></td>
</tr>
<tr>
<td><strong>Arguments:</strong></td>
</tr>
<tr>
<td><strong>Return Value:</strong></td>
</tr>
<tr>
<td><strong>Remarks:</strong></td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
</tr>
<tr>
<td><strong>Coding Example:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>mRtccDisableInt</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Include:</strong></td>
</tr>
<tr>
<td><strong>Prototype:</strong></td>
</tr>
</tbody>
</table>
### mRtccGetIntEnable

**Description:** This macro returns the status of the RTCC interrupts in the INT controller.

**Include:** plib.h

**Prototype:**

```c
int mRtccGetIntEnable (void);
```

**Arguments:** None

**Return Value:** true if the interrupts are enabled, false otherwise

**Remarks:** None

**Source File:** rtcc.h

**Coding Example:**

```c
int isRtccIntEnabled=mRtccGetIntEnable();
```

### mRtccSetIntPriority

**Description:** This macro sets the RTCC event interrupt priority and sub-priority in the interrupt controller.

**Include:** plib.h

**Prototype:**

```c
void mRtccSetIntPriority(int pri, int subPri);
```

**Arguments:**

- pri - the interrupt priority value, 0-7
- subPri - the interrupt sub-priority value, 0-3

**Return Value:** None

**Remarks:** None

**Source File:** rtcc.h

**Coding Example:**

```c
mRtccSetIntPriority(5, 3);
```

### mRtccGetIntPriority

**Description:** This macro returns the RTCC event interrupt priority in the interrupt controller.

**Include:** plib.h

**Prototype:**

```c
void mRtccGetIntPriority(void);
```
19.1.6 Special purpose Functions

These functions control the RTCC operation under special operating conditions, mainly under debugger control. They have no effect under normal operating conditions.

Prototype: int mRtccGetIntPriority(void);
Arguments: None
Return Value: the current RTCC interrupt priority, 0-7
Remarks: None
Source File: rtcc.h
Coding Example: int currPri=mRtccGetIntPriority();

Prototype: int mRtccGetIntSubPriority (void);
Arguments: None
Return Value: the current RTCC interrupt sub-priority, 0-3
Remarks: None
Source File: rtcc.h
Coding Example: int currSubPri= mRtccGetIntSubPriority ();

mRtccFreezeEnable

Description: This macro returns the RTCC event interrupt sub-priority in the interrupt controller.
Include: plib.h
Prototype: int mRtccGetIntSubPriority (void);
Arguments: None
Return Value: the current RTCC interrupt sub-priority, 0-3
Remarks: None
Source File: rtcc.h
Coding Example: int currSubPri= mRtccGetIntSubPriority ();

mRtccGetIntSubPriority

Description: The macro enables the Freeze status of the RTCC.
Include: plib.h
Prototype: void mRtccFreezeEnable (void);
Arguments: None
Return Value: None
Remarks: The Freeze control bit has no significance, unless the processor is under debugger control.
The FRZ bit reads always 0, unless in debug mode.
Source File: rtcc.h
Coding Example: mRtccFreezeEnable ();

19.1.6 Special purpose Functions

These functions control the RTCC operation under special operating conditions, mainly under debugger control. They have no effect under normal operating conditions.
Example of Use

#include <plib.h>

// configuration settings
#pragma config POSCMOD = HS, FNOSC = PRIPLL
#pragma config PLLMUL = MUL_18, PLLIDIV = DIV_2
#pragma config FWDTEN = OFF

int main(void)
{
    rtccTimetm, tm1, tAlrm; // time structure
    rtccDatedt, dt1, dAlrm; // date structure

    // Configure the device for maximum performance.
This macro sets flash wait states, PBCLK divider and DRM wait states based on the specified clock frequency. It also turns on the cache mode if available. Based on the current frequency, the PBCLK divider will be set at 1:2. This knowledge is required to correctly set UART baud rate, timer reload value and other time sensitive setting.

```c
SYSTEMConfigPerformance(72000000L);
RtclInit(); // init the RTCC
while(RtccGetClkStat()!=RTCC_CLK_ON);
// wait for the SOSC to be actually running and RTCC to have
// its clock source. Could wait here at most 32ms

// when using the RtccSetTimeDate() function, the write operation is enabled if needed and
// then restored to the initial value
// so that we don't have to worry about calling RtccWrEnable()/mRtccWrDisable() functions

// let's start setting the current date

// one way to do it
tm.l=0;
tm.sec=0x30;
tm.min=0x07;
tm.hour=0x10;

dt.wday=2;
dt.mday=0x16;
dt.mon=0x01;
dt.year=0x07;
RtccSetTimeDate(tm.l, dt.l);

// however, much easier to do it should be:
RtccSetTimeDate(0x10073000, 0x07011602);
// time is MSb: hour, min, sec, rsvd. date
// date is MSb: year, mon, mday, wday.
// please note that the rsvd field has to be 0 in the time field!

// NOTE: at this point the writes to the RTCC time and date registers are disabled

// we can also read the time and date
tm1.l=RtccGetTime();
dt1.l=RtccGetDate();
// or we can read the time and date in a single operation
RtccGetTimeDate(&tm1, &dt1);
```
// now that we know the RTCC clock is up and running, it's easier to start from fresh:
RtccOpen(tm.l, dt.l, 0); // set time, date and calibration in a single operation

// check that the RTCC is running
{
    int isRunning;
    long retries;
    int secCnt;

    for(secCnt=0; secCnt<3; secCnt++)
    {
        tm.l=RtccGetTime();
        retries=10000000; // how many retries till second changes
        isRunning=0;
        while(retries--)
        {
            tm1.l=RtccGetTime();
            if(tm1.sec!=tm.sec)
            {
                isRunning=1;
                break;
            }
        }

        if(!isRunning)
        {
            break;
        }
    }

    if(isRunning)
    {
        // the RTCC is up and running
    }
}

// let's set the alarm time and check that we actually get an alarm
do
{
    RtccGetTimeDate(&tm, &dt); // get current time and date
}while((tm.sec&0xf)>0x7); // don't want to have minute or BCD rollover

tAlrm.l=tm.l;
dAlrm.l=dt.l;

tAlrm.sec+=2; // alarm due in 2 secs

RtccChimeDisable(); // don't want rollover
RtccSetAlarmRptCount(0); // one alarm will do
RtccSetAlarmRpt(RTCC_RPT_TEN_SEC); // enable repeat rate, check the second field
RtccSetAlarmTimeDate(tAlrm.l, dAlrm.l); // set the alarm time
RtccAlarmEnable(); // enable the alarm

while(RtccGetAlarmEnable()); // wait it to be cleared automatically

// other things we may do with the alarm...
RtccChimeEnable(); // enable indefinite repeats
RtccSetAlarmRptCount(1); // set the initial repeat count
RtccSetAlarmRpt(RTCC_RPT_MIN); // enable repeat rate, every minute, for ex
RtccAlarmDisable(); // disable the alarm
int isAlrmEn=RtccGetAlarmEnable(); // check that the alarm is enabled

// other RTCC operations

// adjust the RTCC timing
RtccSetCalibration(200); // value to calibrate with at each minute

// enabling the RTCC output pin
mRtccSelectSecPulseOutput();
// select the seconds clock pulse as the function of the RTCC output pin
mRtccSelectAlarmPulseOutput(); // select the alarm pulse as the RTCC output pin
mRtccOutputEnable(); // enable the Output pin of the RTCC

// enabling/disabling the RTCC alarm interrupts

// set the RTCC priority and sub-priority in the INT controller
mRtccSetIntPriority(INT_PRIORITY_LEVEL_4, INT_SUB_PRIORITY_LEVEL_1);

mRtccEnableInt(); // enable the RTCC event interrupts in the INT controller.

mRtccDisableInt(); // disable the RTCC interrupts

// once we get in the RTCC ISR we have to clear the RTCC int flag
// but we can do this whenever we see that the interrupt flag is set:
if(mRtccGetIntFlag())
{

mRtccClrIntFlag();
}

// we can check to see if the RTCC interrupts are enabled:
int isRtccIntEn=mRtccGetIntEnable();

return 1;
}
18.0 ADC10 FUNCTIONS

The PIC32MX has an ADC with multiple mode and configuration options. The ADC library functions are available to allow high-level control of the ADC. The following functions and macros are available:

- AcquireADC10() - Starts sample acquisition for the currently select channel
- BusyADC10() - Returns the status of the conversion done bit.
- CloseADC10() - Disables and turns off the ADC.
- ConfigIntADC10() - Configures the priority and sub-priority for the ADC interrupt and enables the interrupt.
- ConvertADC10() - Starts a conversion for the acquired sample.
- EnableADC10() - Turns the ADC on
- OpenADC10() - Configures and enables the ADC module.
- ReadActiveBufferADC10() - Returns the buffer that is being written when Dual Buffer mode is in use
- ReadADC10() - Returns the vaule in the specified location of the ADC result buffer.
- SetChanADC10() - Configures the ADC input multiplexers

18.1 Individual Functions

There are no functions to support this module, refer to the macro section

18.2 Individual Macros

### AcquireADC10

**Description:** This function starts A/D acquisition when the ADC is in manual conversion and manual sample mode.

**Include:** plib.h  
**Prototype:** AcquireADC10();  
**Arguments:** None  
**Return Value:** None  
**Remarks:** This macro sets the ADCON1<SAMP> bit and thus starts sampling. This happens only when trigger source for the A/D conversion is selected as Manual, by clearing the ADCON1 <SSRC> bits.

**Code Example:** ConvertADC10();

### BusyADC10

**Description:** This macro returns the ADC conversion status.

**Include:** plib.h
BusyADC10 (Continued)

Prototype: int BusyADC10();
Arguments: None
Return Value: ‘1’ if ADC is busy in conversion.
‘0’ if ADC is has completed conversion or currently not performing any conversion.
Remarks: None
Code Example: while(BusyADC10());

CloseADC10

Description: This macro turns off the ADC module and disables the ADC interrupts.
Include: plib.h
Prototype: CloseADC10();
Arguments: None
Return Value: None
Remarks: This function first disables the ADC interrupt and then turns off the ADC module. The Interrupt Flag bit (ADIF) is also cleared.
Code Example: CloseADC10();

ConfigIntADC10

Description: This function configures the ADC interrupt.
Include: plib.h
Prototype: ConfigIntADC10(unsigned long int config);
Arguments: config This contains the bit fields that make up the parameter. A logical OR is used to combine multiple bit fields together.

<table>
<thead>
<tr>
<th>ADC Interrupt enable/disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_INT_ENABLE</td>
</tr>
<tr>
<td>ADC_INT_DISABLE</td>
</tr>
<tr>
<td>(These bit fields are mutually exclusive)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADC Interrupt priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_INT_PRI_0</td>
</tr>
<tr>
<td>ADC_INT_PRI_1</td>
</tr>
<tr>
<td>ADC_INT_PRI_2</td>
</tr>
<tr>
<td>ADC_INT_PRI_3</td>
</tr>
<tr>
<td>ADC_INT_PRI_4</td>
</tr>
<tr>
<td>ADC_INT_PRI_5</td>
</tr>
<tr>
<td>ADC_INT_PRI_6</td>
</tr>
<tr>
<td>ADC_INT_PRI_7</td>
</tr>
<tr>
<td>(These bit fields are mutually exclusive)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADC Interrupt sub priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_SUB_INT_PRI_0</td>
</tr>
<tr>
<td>ADC_SUB_INT_PRI_1</td>
</tr>
<tr>
<td>ADC_SUB_INT_PRI_2</td>
</tr>
<tr>
<td>ADC_SUB_INT_PRI_3</td>
</tr>
<tr>
<td>(These bit fields are mutually exclusive)</td>
</tr>
</tbody>
</table>
ConfigIntADC10

Remarks: This function clears the Interrupt Flag (ADIF) bit and then sets the interrupt priority and enables/disables the interrupt.

Code Example: ConfigIntADC10(ADC_INT_PRI_3 | ADC_INT_SUB_PRI_3 | ADC_INT_ENABLE);

ConvertADC10

Description: This function starts the A/D conversion when the AC is in manual conversion mode.

Include: plib.h

Prototype: ConvertADC10();

Arguments: None

Return Value: None

Remarks: This function clears the ADCON1<SAMP> bit and thus stops sampling and starts conversion.

Code Example: ConvertADC10();

EnableADC10

Description: This macro enables the ADC.

Include: plib.h

Prototype: EnableADC10();

Arguments: None

Remarks: This macro is intended for use when the ADC is configured but not enabled by prior operations. The ADC configuration should not be changed while the ADC is enabled.

Code Example: EnableADC10();

OpenADC10

Description: This function configures the ADC using the 5 parameters passed to it.

Include: plib.h

Prototype: void OpenADC10(unsigned long int config1,
                              unsigned long int config2,
                              unsigned long int config3,
                              unsigned long int configport,
                              unsigned long int configscan)

Arguments: config1 This contains the bit fields that make up the parameter for the AD1CON1 register. A logical OR is used to combine multiple bit fields together.

Module On/Off
ADC_MODULE_ON
ADC_MODULE_OFF
(These bit fields are mutually exclusive)
<table>
<thead>
<tr>
<th>config2</th>
<th>This contains the bit fields that make up the parameter for the AD1CON2 register. A logical OR is used to combine multiple bit fields together.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OpenADC10</strong></td>
<td></td>
</tr>
</tbody>
</table>
| **Idle mode operation** | ADC_IDLE_CONTINUE  
ADC_IDLE_STOP  
(These bit fields are mutually exclusive) |
| **Result output format (16 bit justified)** | ADC_FORMAT_SIGN_FRACT16  
ADC_FORMAT_FRACT16  
ADC_FORMAT_SIGN_INT16  
ADC_FORMAT_INTG16 |
| **Result output format (32 bit justified)** | ADC_FORMAT_SIGN_FRACT32  
ADC_FORMAT_FRACT32  
ADC_FORMAT_SIGN_INT32  
ADC_FORMAT_INTG32  
(These bit fields are mutually exclusive) |
| **Conversion trigger source** | ADC_CLK_AUTO  
ADC_CLK_TMR  
ADC_CLK_INT0  
ADC_CLK_MANUAL  
(These bit fields are mutually exclusive) |
| **Auto sampling select** | ADC_AUTO_SAMPLING_ON  
ADC_AUTO_SAMPLING_OFF  
(These bit fields are mutually exclusive) |
| **Sample enable** | ADC_SAMP_ON  
ADC_SAMP_OFF  
(These bit fields are mutually exclusive) |
| **Voltage Reference** | ADC_VREF_AVDD_AVSS  
ADC_VREF_EXT_AVSS  
ADC_VREF_AVDD_EXT  
ADC_VREF_EXT_EXT  
(These bit fields are mutually exclusive) |
| **Offset Calibration Mode** | ADC_OFFSET_CAL_ENABLE  
ADC_OFFSET_CAL_DISABLE  
(These bit fields are mutually exclusive) |
| **Scan selection** | ADC_SCAN_ON  
ADC_SCAN_OFF  
(These bit fields are mutually exclusive) |
## OpenADC10

<table>
<thead>
<tr>
<th>Number of samples between interrupts</th>
<th>ADC_SAMPLES_PER_INT_1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADC_SAMPLES_PER_INT_2</td>
</tr>
<tr>
<td></td>
<td>.....</td>
</tr>
<tr>
<td></td>
<td>ADC_SAMPLES_PER_INT_15</td>
</tr>
<tr>
<td></td>
<td>ADC_SAMPLES_PER_INT_16</td>
</tr>
</tbody>
</table>

(These bit fields are mutually exclusive)

<table>
<thead>
<tr>
<th>Buffer mode select</th>
<th>ADC_ALT_BUF_ON</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADC_ALT_BUF_OFF</td>
</tr>
</tbody>
</table>

(These bit fields are mutually exclusive)

<table>
<thead>
<tr>
<th>Alternate Input Sample mode select</th>
<th>ADC_ALT_INPUT_ON</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADC_ALT_INPUT_OFF</td>
</tr>
</tbody>
</table>

(These bit fields are mutually exclusive)

| config3                           | This contains the bit fields that make up the parameter for the AD1CON3 register. A logical OR is used to combine multiple bit fields together. |

<table>
<thead>
<tr>
<th>Auto Sample Time bits</th>
<th>ADC_SAMPLE_TIME_0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADC_SAMPLE_TIME_1</td>
</tr>
<tr>
<td></td>
<td>.....</td>
</tr>
<tr>
<td></td>
<td>ADC_SAMPLE_TIME_30</td>
</tr>
<tr>
<td></td>
<td>ADC_SAMPLE_TIME_31</td>
</tr>
</tbody>
</table>

(These bit fields are mutually exclusive)

<table>
<thead>
<tr>
<th>Conversion Clock Source select</th>
<th>ADC_CONV_CLK_INTERNAL_RC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADC_CONV_CLK_SYSTEM</td>
</tr>
</tbody>
</table>

(These bit fields are mutually exclusive)

<table>
<thead>
<tr>
<th>Conversion clock select</th>
<th>ADC_CONV_CLK_Tcy2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADC_CONV_CLK_Tcy</td>
</tr>
<tr>
<td></td>
<td>ADC_CONV_CLK_3Tcy2</td>
</tr>
<tr>
<td></td>
<td>.....</td>
</tr>
<tr>
<td></td>
<td>ADC_CONV_CLK_32Tcy</td>
</tr>
</tbody>
</table>

(These bit fields are mutually exclusive)
OpenADC10

configport  This contains the bit fields that make up the parameter for the AD1PCFG register. A logical OR is used to combine multiple bit fields together.

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE_ALL_ANA</td>
</tr>
<tr>
<td>ENABLE_ALL_DIG</td>
</tr>
<tr>
<td>ENABLE_AN0_ANA</td>
</tr>
<tr>
<td>ENABLE_AN1_ANA</td>
</tr>
<tr>
<td>ENABLE_AN2_ANA</td>
</tr>
<tr>
<td>....</td>
</tr>
<tr>
<td>ENABLE_AN15_ANA</td>
</tr>
</tbody>
</table>

configscan  This contains the bit fields that make up the parameter for the AD1CSSL register. A logical OR is used to combine multiple bit fields together.

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SCAN_SCAN_NONE</td>
</tr>
<tr>
<td>SCAN_SCAN_ALL</td>
</tr>
<tr>
<td>SKIP_SCAN_AN0</td>
</tr>
<tr>
<td>SKIP_SCAN_AN1</td>
</tr>
<tr>
<td>....</td>
</tr>
<tr>
<td>SKIP_SCAN_AN15</td>
</tr>
</tbody>
</table>

Return Value: None

Remarks: This function configures the ADC for the following parameters: Operating mode, Sleep mode behavior, Data output format, Sample Clk Source, VREF source, No of samples/int, Buffer Fill mode, Alternate input sample mod, Auto sample time, Conv clock source, Conv Clock Select bits, Port Config Control bits. Channel select for manual and alternate sample modes is not configured by this macro.

Code Example:

```c
OpenADC10(ADC_MODULE_OFF | ADC_IDLE_STOP | ADC_FORMAT_SIGN_FRACT16 | ADC_CLK_INT0 | ADC_SAMPLE_INDIVIDUAL | ADC_AUTO_SAMPLING_ON, ADC_VREF_AVDD_AVSS | ADC_SCAN_OFF | ADC_ALT_INPUT_ON | ADC_SAMPLES_PER_INT_10, ADC_SAMPLE_TIME_4 | ADC_CONV_CLK_PB | ADC_CONV_CLK_Tcy, ENABLE_AN1_ANA, SKIP_SCAN_AN0 | SKIP_SCAN_AN3 | SKIP_SCAN_AN4 | SKIP_SCAN_AN5);
```

ReadActiveBufferADC10

Description: This macro returns the status of the buffer fill bit.

Include: plib.h

Prototype: ReadActiveBufferADC10();

Arguments: None
Remarks: This macro is intended for use when the ADC output buffer is used in dual buffer mode. A ‘0’ result indicates that buffer locations 0-7 are being written by the ADC module. A ‘1’ result indicates that buffer locations 8-F are being written by the ADC module.

Code Example:

```c
unsigned long int a;
a = ReadActiveBufferADC10();
```
### ReadADC10

**Description:** This function reads the specified entry in the ADC result buffer which contains the conversion value.

**Include:** `plib.h`

**Prototype:**
```
ReadADC10(unsigned long int bufIndex);
```

**Arguments:**
- `bufIndex` This is the ADC buffer number which is to be read.

**Return Value:** The corresponding entry from the ADC result buffer

**Remarks:** This function returns the contents of the ADC Buffer register. User should provide `bufIndex` value between ’0’ to ’15’ to ensure a correct read of AD1CBUF0 through AD1CBUFF.

**Code Example:**
```
unsigned long int result;
result = ReadADC10(3);
```

### SetChanADC10

**Description:** This function sets the positive and negative inputs for the sample multiplexers A and B for manual and alternate sample modes.

**Include:** `plib.h`

**Prototype:**
```
SetChanADC10(unsigned int channel);
```

**Arguments:**
- `channel` This contains the bit fields that make up the parameter. A logical OR is used to combine multiple bit fields together.

<table>
<thead>
<tr>
<th>A/D Channel 0 positive input select for Sample A</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_CH0_POS_SAMPLEA_AN0</td>
</tr>
<tr>
<td>ADC_CH0_POS_SAMPLEA_AN1</td>
</tr>
<tr>
<td>.....</td>
</tr>
<tr>
<td>ADC_CH0_POS_SAMPLEA_AN15</td>
</tr>
<tr>
<td>(These bit fields are mutually exclusive)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A/D Channel 0 negative input select for Sample A</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_CH0_NEG_SAMPLEA_AN1</td>
</tr>
<tr>
<td>ADC_CH0_NEG_SAMPLEA_NVREF</td>
</tr>
<tr>
<td>(These bit fields are mutually exclusive)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A/D Channel 0 positive input select for Sample B</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_CH0_POS_SAMPLEB_AN0</td>
</tr>
<tr>
<td>ADC_CH0_POS_SAMPLEB_AN1</td>
</tr>
<tr>
<td>.....</td>
</tr>
<tr>
<td>ADC_CH0_POS_SAMPLEB_AN15</td>
</tr>
<tr>
<td>(These bit fields are mutually exclusive)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A/D Channel 0 negative input select for Sample B</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_CH0_NEG_SAMPLEB_AN1</td>
</tr>
<tr>
<td>ADC_CH0_NEG_SAMPLEB_NVREF</td>
</tr>
<tr>
<td>(These bit fields are mutually exclusive)</td>
</tr>
</tbody>
</table>

**Return Value:** None

**Remarks:** This function configures the inputs for sample multiplexers A and B by writing to ADCHS register. This macro is intended for use when configuring the positive inputs when not using scan mode. This macro can be used to configure the negative input for the ADC in all modes of operation.
SetChanADC10 (Continued)

Code Example:

```
SetChanADC10(ADC_CH0_POS_SAMPLEA_AN0 |
           ADC_CH0_NEG_SAMPLEA_NVREF);
```

18.3 Example of Use

// Master header file for all peripheral library includes
#include <plib.h>

unsigned int channel4; // conversion result as read from result buffer
unsigned int channel5; // conversion result as read from result buffer
unsigned int offset;   // points to the base of the idle buffer

main()
{
  // configure and enable the ADC
  CloseADC10(); // ensure the ADC is off before setting the configuration

  // define setup parameters for OpenADC10
  #define PARAM1   ADC_MODULE_ON | ADC_FORMAT_INTG | ADC_CLK_AUTO | 
                 ADC_AUTO_SAMPLING_ON

  #define PARAM2   ADC_VREF_AVDD_AVSS | ADC_OFFSET_CAL_DISABLE | ADC_SCAN_OFF | 
                 ADC_SAMPLES_PER_INT_2 | ADC_ALT_BUF_ON | ADC_ALT_INPUT_ON

  #define PARAM3   ADC_CONV_CLK_INTERNAL_RC | ADC_SAMPLE_TIME_12

  #define PARAM4   SKIP_SCAN_ALL

  #define PARAM5   ENABLE_AN4_ANA | ENABLE_AN5_ANA

  // configure to sample AN4 & AN5
  SetChanADC10( ADC_CH0_NEG_SAMPLEA_NVREF | ADC_CH0_POS_SAMPLEA_AN4 | 
                 ADC_CH0_NEG_SAMPLEB_NVREF | ADC_CH0_POS_SAMPLEB_AN5);

  // configure ADC and enable it
  OpenADC10( PARAM1, PARAM2, PARAM3, PARAM4, PARAM5 );

  // Now enable the ADC logic
  EnableADC10();

  // the results of the conversions are available in channel4 and channel5
  while (1)
  {
    // determine which buffer is idle and create an offset
    offset = 8 * ((~ReadActiveBufferADC10() & 0x01));

    // read the result of channel 4 conversion in the idle buffer
    channel4 = ReadADC10(offset);

    // read the result of channel 5 conversion in the idle buffer
    channel5 = ReadADC10(offset + 1);
  }
}
19.0 COMPARATOR FUNCTIONS

The PIC32MX has analog comparators with multiple configuration options. The comparator library functions are available to allow high-level control of the comparators. The following macros are available:

CMP1Close(), CMP2Close() - Disables the comparators interrupt and turns off both comparators.
CMP1ConfigInt(), CMP2ConfigInt() - Configures the interrupt for the comparator.
CMP1Open(), CMP2Open() - Configures the comparator inputs, and event generation.
CMP1Read(), CMP2Read() - Reads the status of the comparator output bit.

19.1 Individual Functions

There are no functions to support this module, refer to the macro section

19.2 Individual Macros

CMP1Close()
CMP2Close()

Description: This macro disables the Comparator module.
Include: plib.h
Prototype: CMP1Close();
          CMP2Close();
Arguments: None
Return Value: None
Remarks: This function turns the CMP module off and disables the interrupt.
Code Example: CMP1Close();

CMP1Open()
CMP2Open()

Description: This macro configures and turns on the comparator module.
Include: plib.h
Prototype: CMP1Open(unsigned long int config);
          CMP2Open(unsigned long int config);
Arguments: config This contains the input select parameter to be written into
          the CVRCON register as defined below:
          CMP Mode Select
          CMP_ENABLE
          CMP_DISABLE
          (These bit fields are mutually exclusive)
CMP1Open() (Continued)

CMP2Open() (Continued)

<table>
<thead>
<tr>
<th>CMP Operation In Idle</th>
<th>CMP_RUN_IN_IDLE</th>
<th>CMP_HALT_IN_IDLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(These bit fields are mutually exclusive)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMP Output Control</th>
<th>CMP_OUTPUT_ENABLE</th>
<th>CMP_OUTPUT_DISABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(These bit fields are mutually exclusive)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMP Polarity Select</th>
<th>CMP_OUTPUT_INVERT</th>
<th>CMP_OUTPUT_NONINVERT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(These bit fields are mutually exclusive)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMP Interrupt Event Select</th>
<th>CMP_EVENT_NONE</th>
<th>CMP_EVENT_LOW_TO_HIGH</th>
<th>CMP_EVENT_HIGH_TO_LOW</th>
<th>CMP_EVENT_CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(These bit fields are mutually exclusive)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMP1 Positive Input Select</th>
<th>CMP_POS_INPUT_C1IN_POS</th>
<th>CMP_POS_INPUT_CVREF</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Only use these bit fields for CMP1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(These bit fields are mutually exclusive)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMP2 Positive Input Select</th>
<th>CMP_POS_INPUT_C2IN_POS0</th>
<th>CMP_POS_INPUT_CVREF</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Only use these bit fields for CMP2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(These bit fields are mutually exclusive)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMP1 Negative Input Select</th>
<th>CMP1_NEG_INPUT_C1IN_NEG</th>
<th>CMP1_NEG_INPUT_C1IN_POS</th>
<th>CMP1_NEG_INPUT_C2IN_POS</th>
<th>CMP1_NEG_INPUT_IVREF</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Only use these bit fields for CMP1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(These bit fields are mutually exclusive)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CMP2 Negative Input Select</th>
<th>CMP2_NEG_INPUT_C2IN_NEG</th>
<th>CMP2_NEG_INPUT_C2IN_POS</th>
<th>CMP2_NEG_INPUT_C1IN_POS</th>
<th>CMP2_NEG_INPUT_IVREF</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Only use these bit fields for CMP2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(These bit fields are mutually exclusive)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Value: None

Remarks: The Stop in Idle function is common to both comparators. Therefore both comparators will have their Idle mode behavior set by the last CMxOpen macro used.

Code Example: CMP1Open(CMP_ENABLE | CMP_HALT_IN_IDLE | CMP_OUTPUT_ENABLE | CMP_OUTPUT_INVERT | CMP_EVENT_LOW_TO_HIGH | CMP_POS_INPUT_CVREF | CMP_NEG_INPUT_CINC);
CMP1IntConfig()
CMP2IntConfig()

Description: This function configures comparator interrupt priority and sub-priority values.

Include: plib.h

Prototype: CMP1IntConfig();
            CMP2IntConfig();

Arguments: config This contains the input select parameter to configure interrupt setting.: A bit-wise OR is used to combine multiple bit fields together.

    CMP Interrupt Control
    CMP_INT_ENABLE
    CMP_INT_DISABLE
    (These bit fields are mutually exclusive)
    CMP_INT_SUB_PRIORITY0
    CMP_INT_SUB_PRIORITY1
    CMP_INT_SUB_PRIORITY2
    CMP_INT_SUB_PRIORITY3
    (These bit fields are mutually exclusive)

Return Value:    CMP Interrupt Priority
    CMP_INT_PRIORITY0
    CMP_INT_PRIORITY1
    CMP_INT_PRIORITY2
    CMP_INT_PRIORITY3
    CMP_INT_PRIORITY4
    CMP_INT_PRIORITY5
    CMP_INT_PRIORITY6
    CMP_INT_PRIORITY7
    (These bit fields are mutually exclusive)

Remarks:  Code Example:
          unsigned long int result;
          result = CMP1IntConfig(CMP_INT_ENABLE |
                                   CMP_INT_PRIORITY3 | CMP_INT_SUB_PRIORITY2);

CMP1Read()
CMP2Read()

Description: This function reads the status of the comparator output bit.

Include: plib.h

Prototype: CMP1Read();
            CMP2Read();

Arguments: None

Return Value: None

Remarks:  Code Example:
          unsigned long int result;
          result = CMP1Read();
19.3 Example of Use

```c
#include <plib.h>

int main(void)
{
    unsigned int status;

    // Configure comparator 1
    CMP1Open( CMP_ENABLE | CMP_OUTPUT_NONINVERT | CMP_EVENT_NONE |
               CMP_POS_INPUT_C1IN_POS | CMP1_NEG_INPUT_C1IN_NEG );

    while ( 1 )
    {
        status = CMP1Read(); // get the current status of the comparator
    }

    CMP1Close(); // note: not executed

    return 0;
}
```
20.0 CVREF FUNCTIONS

The PIC32MX has comparator voltage reference with multiple configuration options. The CVREF library functions are available to allow high-level control of the module. The following macros are available:

CVREFClose() - Disables the CVREF module and disable the output pin.
CVREFOpen() - Enables the CVREF module. Sets the output voltage, configure the output range, and configures the output to a pin.

20.1 Individual Functions
There are no functions to support this module, refer to the macro section

20.2 Individual Macros

### CVREFClose()

<table>
<thead>
<tr>
<th>Description</th>
<th>This macro disables the CVREF module.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>plib.h</td>
</tr>
<tr>
<td>Prototype</td>
<td>CVREFClose();</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
<tr>
<td>Remarks</td>
<td>This function turns the CVREF module off and disables the output.</td>
</tr>
<tr>
<td>Code Example</td>
<td>CVREFClose();</td>
</tr>
</tbody>
</table>

### CVREFOpen()

<table>
<thead>
<tr>
<th>Description</th>
<th>This macro configures and turns on the CVREF module.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>plib.h</td>
</tr>
<tr>
<td>Prototype</td>
<td>void CVREFOpen(unsigned int config);</td>
</tr>
<tr>
<td>Arguments</td>
<td>config This contains the bit fields that make up the parameter. A logical OR is used to combine multiple bit fields together.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CVREF Mode Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVREF_ENABLE</td>
</tr>
<tr>
<td>CVREF_DISABLE</td>
</tr>
<tr>
<td>(These bit fields are mutually exclusive)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CVREF Output Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVREF_OUTPUT_ENABLE</td>
</tr>
<tr>
<td>CVREF_OUTPUT_DISABLE</td>
</tr>
<tr>
<td>(These bit fields are mutually exclusive)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CVREF Range Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVREF_RANGE_HIGH</td>
</tr>
<tr>
<td>CVREF_RANGE_LOW</td>
</tr>
<tr>
<td>(These bit fields are mutually exclusive)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CVREF Reference Source Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVREF_SOURCE_AVDD</td>
</tr>
<tr>
<td>CVREF_SOURCE_VREF</td>
</tr>
<tr>
<td>(These bit fields are mutually exclusive)</td>
</tr>
</tbody>
</table>
CVREFOpen() (Continued)

<table>
<thead>
<tr>
<th>CVREF Output Voltage Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVREF_STEP_0</td>
</tr>
<tr>
<td>CVREF_STEP_1</td>
</tr>
<tr>
<td>CVREF_STEP_2</td>
</tr>
<tr>
<td>CVREF_STEP_3</td>
</tr>
<tr>
<td>CVREF_STEP_4</td>
</tr>
<tr>
<td>CVREF_STEP_5</td>
</tr>
<tr>
<td>CVREF_STEP_6</td>
</tr>
<tr>
<td>CVREF_STEP_7</td>
</tr>
<tr>
<td>CVREF_STEP_8</td>
</tr>
<tr>
<td>CVREF_STEP_9</td>
</tr>
<tr>
<td>CVREF_STEP_10</td>
</tr>
<tr>
<td>CVREF_STEP_11</td>
</tr>
<tr>
<td>CVREF_STEP_12</td>
</tr>
<tr>
<td>CVREF_STEP_13</td>
</tr>
<tr>
<td>CVREF_STEP_14</td>
</tr>
<tr>
<td>CVREF_STEP_15</td>
</tr>
</tbody>
</table>

(These bit fields are mutually exclusive)

Return Value: None
Remarks: 
Code Example: CVREFOpen(CVREF_ENABLE | CVREF_OUTPUT_ENABLE 
CVREF_RANGE_HIGH | CVREF_SOURCE_AVDD | 
CVREF_STEP_15);
20.3 Example of Use

// Master header file for all peripheral library includes
#include <plib.h>

// this program generates an approximation of a triangle wave
main()
{
    unsigned int step;
    unsigned int loop;
    unsigned int ramp;

    while(1)
    {
        for ( loop =0; loop <= 15; loop ++)
        {
            for ( ramp = 0;  ramp <= 31; ramp ++)
            {
                if (  ramp <= 15 )
                    step = ramp;
                else
                    step = 31 - ramp;

                CVREFOpen( CVREF_ENABLE | CVREF_OUTPUT_ENABLE | CVREF_RANGE_HIGH
                          | CVREF_SOURCE_AVDD | step );
            }
        }
        CVREFClose(); // Disable CVREF (not executed)
    }
}
## 21.0 WDT FUNCTIONS

This section contains a list of individual functions for the WatchDog Timer and an example of use of the functions. Functions may be implemented as macros.

### 21.1 Individual Functions

There are no functions to support this module, refer to the macro section

### 21.2 Individual Macros

#### DisableWDT()

<table>
<thead>
<tr>
<th>Description:</th>
<th>This function disables the WDT.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>plib.h</td>
</tr>
<tr>
<td>Arguments:</td>
<td>None</td>
</tr>
<tr>
<td>Prototype:</td>
<td>void DisableWDT(void);</td>
</tr>
<tr>
<td>Return Value:</td>
<td>None</td>
</tr>
</tbody>
</table>

#### EnableWDT()

<table>
<thead>
<tr>
<th>Description:</th>
<th>This function enables the WDT.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>plib.h</td>
</tr>
<tr>
<td>Arguments:</td>
<td>Mode This contains the bit fields that make up the parameter.</td>
</tr>
<tr>
<td>Prototype:</td>
<td>void EnableWDT(void);</td>
</tr>
<tr>
<td>Return Value:</td>
<td>None</td>
</tr>
</tbody>
</table>

#### ClearWDT()

<table>
<thead>
<tr>
<th>Description:</th>
<th>This function resets the WDT timer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include:</td>
<td>plib.h</td>
</tr>
<tr>
<td>Arguments:</td>
<td>None</td>
</tr>
<tr>
<td>Prototype:</td>
<td>void ClearWDT(void);</td>
</tr>
<tr>
<td>Return Value:</td>
<td>None</td>
</tr>
</tbody>
</table>

Remarks: The WDT can only be disabled in software if it was not enabled by the WDT fuse.

Code Example: DisableWDT();

Remarks: This function can be used to enable the wdt module.

Code Example: EnableWDT();

Remarks: This function has no effect if the WDT is not enabled.

Code Example: ClearWDT();
### ClearEventWDT()

**Description:** This function clears the WDT event bit.

**Include:** plib.h

**Arguments:** None

**Prototype:**

```c
void ClearEventWDT(void);
```

**Return Value:** None

**Remarks:** This function allows the WDT event bit to be reset after the startup code has determined the source of the device reset.

**Code Example:**

```c
ClearEventWDT();
```

---

### ReadEventWDT()

**Description:** This function reads the status of the WDT event bit.

**Include:** plib.h

**Arguments:** None

**Prototype:**

```c
unsigned int ReadEventWDT(void);
```

**Return Value:** The status of the WDT event bit

**Remarks:**

**Code Example:**

```c
unsigned int eventBitWDT;
eventBitWDT = ReadEventWDT();
```

---

### ReadPostscalerWDT()

**Description:** This function reads the value of the WDT postscaler

**Include:** plib.h

**Arguments:** None

**Prototype:**

```c
unsigned int ReadPostscalerWDT(void);
```

**Return Value:** The value of the WDT Postscaler

**Remarks:**

**Code Example:**

```c
unsigned int postscalerValue;
postscalerValue = ReadPostscalerWDT();
```