I^2C Master Datasheet I2Cm V 2.00

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/24/22/21xxx, CY7C603xx, CY7C64215, CYWUSB6953, CY8C23x33, CY8CLED0xD, CY8CLED0xG, CY8CLED02/04/08/16, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx, CY8C21x12						
	0	0	0	791	4	2

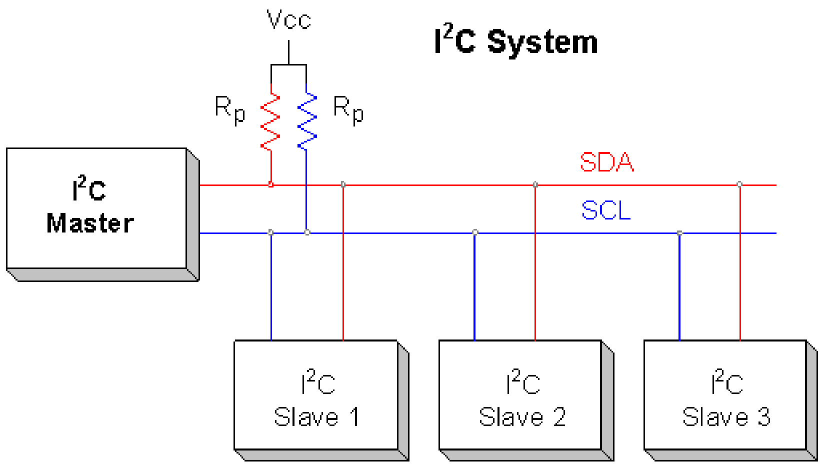
Features and Overview

- Industry standard Philips I^2C bus compatible interface
- Only two pins (SDA and SCL) required to interface several slave I^2C devices
- Standard mode data supports rate of 100 kbps
- High level API requires minimal user programming
- Low level API provided for flexibility

The I^2Cm User Module implements a master I^2C device in firmware. The I^2C bus is an industry standard, two-wire interface developed by Philips®. An I^2C bus master may communicate with several slave devices using only two wires. The master initiates all communication on the I^2C bus and supplies the clock for all slave devices. The I^2Cm User Module supports speeds up to 100 kbps. No digital or analog user blocks are consumed with this module.

The Application Programming Interface (API) firmware provides high-level commands that support sending and receiving multiple bytes with a single function call. Repeat starts are supported, but multi-master arbitration and 10-bit addressing are not.

Figure 1. I^2C Block Diagram



Functional Description

This user module implements an I^2C master in firmware. It is capable of data transfer rates up to 100 kbps when the CPU clock is configured to run at 24 MHz. Slower CPU clocks may be used, but the data transfer rate will slow down accordingly. The I^2C specification allows the master to run at clock speeds from 100 kHz down to DC. Any two pins on a single port may be selected to drive the SDA and SCL signals.

This module does not require any analog or digital PSoC blocks and therefore it does not use interrupts. When a data transfer occurs, the CPU is 100 percent utilized. Background interrupts do not have to be disabled during transfers, since the I^2C bus specification allows the bus clock to operate between DC and 100 kHz in the standard mode. Only the 7-bit address mode is supported.

The pull up resistors (R_P) are determined by the supply voltage, clock speed, and bus capacitance. The minimum sink current for any device (master or slave) should be no less than 3 mA at V_{OLmax} = 0.4V for the output stage. This limits the minimum pull up resistor value for a 5 volt system to about 1.5K ohms. The maximum value for R_P is dependent on the bus capacitance and the clock speed. For a 5 volt system with a bus capacitance of 150 pF, the pull up resistors should be no larger than 6K ohms. For more information on “The I^2C-Bus Specification", see the Philips web site at www.philips.com.

I^2C addresses are contained in the upper 7 bits of the address byte. Valid selections are from 0-127(dec). The LSB of the byte contains the R/~W bit. If this bit is 0, the address will be written to, if the LSB is a 1 then the addressed slave will have data read from it.

Internally the user module will take the input address, shift it and combine it with a read/write bit to construct a compete address byte.

For example, an address of 0x48 is passed as a parameter. A separate parameter is passed containing read/write information. An I^2C Master would send a byte (8-bits) of 0x90 to write data to the slave and the byte 0x91 to read data from the slave.

Direct writes to the PORT used by the I2Cm User Module will interfere with I^2C operation. A shadow register is automatically added for the port used by the I2Cm User Module. For example, if Port1 is used by the module, a shadow register named Port_1_DriveMode_0_SHADE is added to the project. All writes to Port1 should be done using this shadow register. For example, if Port1 bits 5 and 7 are used by the I2Cm User Module, a modification to port pin 6 could be made as follows:

C code:

```
Port_1_DriveMode_0_SHADE ^= 0x40;
PRT1DR = Port_1_DriveMode_0_SHADE;
```

Assembly code:

```
xor [Port_1_DriveMode_0_SHADE], 0x40
mov A, [Port_1_DriveMode_0_SHADE]
mov reg[PRT1DR], A
```

Note Purchase of I^2C components from Cypress, or one of its sublicensed Associated Companies, conveys a license under the Philips I^2C Patent Rights to use these components in an I^2C-bus system, provided that the system conforms to the I^2C Standard Specification as defined by Philips.

Placement

The I^2C Master User Module uses two I/O pins of one port and does not require digital or analog PSoC blocks. There are no placement restrictions. Multiple I^2C master modules may be placed in a single project, although multi-master mode is not supported on a single bus.

Parameters and Resources

I2C_Port

Selects which PSoC I/O port is used to interface the SDA and SCL signals.

SDA_Pin

Selects which pin of the I2C_Port, the SDA data signal will be present. There is no need to select the proper drive mode for this pin; PSoC Designer will do this automatically.

SCL_Pin

Selects which pin of the I2C_Port, the SCL Clock signal will be present. There is no need to select the proper drive mode for this pin; PSoC designer will do this automatically.

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the “include" files.

Note

In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X prior to the call if those values are required after the call. This “registers are volatile" policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API function may leave A and X unchanged, there is no guarantee they will do so in the future.

Two different methods of passing API parameters are supported within the provided APIs. Early versions of the parameter passing method were obsolete with later C-compiler versions. The impact of this change would only be felt by customers using assembly code and the old API calling structure with a small memory model device if they were upgrading their application to a large memory model device. Newer applications may use the calling structures described below (new style parameter passing) with assembly language implementations by adding an underscore to the assembly call statement. No applications written in C are affected. Routines that fit this description are: I2Cm_fReadBytes, I2Cm_bWriteBytes, and I2Cm_bWriteCBytes.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

I2Cm_Start

Description:

Initializes the I/O mode and initial levels for the pins selected to function as SDA and SCL.

C Prototype:

```
void I2Cm_Start(void);
```

Assembler:

```
lcall I2Cm_Start
```

Parameters:

None

Return Value:

None

Side Effects:

You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

I2Cm_Stop

Description:

This function performs no function, but is implemented for future compatibility.

C Prototype:

```
void I2Cm_Stop(void);
```

Assembler:

```
lcall I2Cm_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

I2Cm_fReadBytes

Description:

Reads one or more bytes (bCnt) from the slave I^2C device and writes data to the array pointed to by pbXferData.

C Prototype:

```
BooL I2Cm_fReadBytes(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE bMode);
```

Assembler:

```
mov   A,I2Cm_CompleteXfer    ; Pass complete transfer flag
push  A
mov   A,0x09                 ; Pass the byte count
push  A
mov   A,>sData               ; Load the MSB of the sData pointer
push  A
mov   A,<sData               ; Load the LSB of the sData pointer
push  A
mov   X,SP                   ; Get the stack location +1
dec   X                      ; X now points to the last byte pushed
mov   A,0x68                 ; Pass slave address 0x68
lcall  _I2Cm_fReadBytes       ; Call function to read data from slave
                             ; Reg A contains return value.
add   sp,-4                  ; Restore the stack
```

Parameters:

bSlaveAddr: 7-bit slave address.

pbXferData: Pointer to data array in RAM.

bCnt: Count of data to read.

bMode: Mode of operation. If mode is set to I2Cm_CompleteXfer, a complete transfer is performed. If mode is set to I2Cm_RepStart, a repeat start condition will be generated instead of a start condition. If mode is set to I2Cm_NoStop, a stop is not generated. This allows an I^{2}C bus combined transfer to be sent to the slave. See the table at the end of this section.

Return Value:

If transfer is completed without errors, a non-zero value is returned. If transfer has failed, a zero is returned.

Side Effects:

You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, the CUR_PP and IDX_PP page pointer registers are modified.

I2Cm_bWriteBytes

Description:

Writes one or more bytes (bCnt) to the slave addressed by (bSlaveAddr) from the RAM array pointed to by (pbXferData).

C Prototype:

```
BYTE I2Cm_bWriteBytes(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE bMode);
```

Assembler:

```
mov   A,I2Cm_CompleteXfer    ; Pass complete transfer flag
push  A
mov   A,0x09                 ; Pass the byte count
push  A
```

```
mov   A,>sData               ; Load the MSB of the sData pointer
push  A
mov   A,<sData               ; Load the LSB of the sData pointer
push  A
mov   X,SP                   ; Get the stack location +1
dec   X                      ; X now points to the last byte pushed
mov   A,0x68                 ; Pass slave address 0x68
lcall  _I2Cm_bWriteBytes      ; Call function to write data to slave
                             ; Reg A contains return value.
add   sp,-4                  ; Restore the stack
```

Parameters:

bSlaveAddr: 7-bit slave address.

pbXferData: Pointer to data array in RAM.

bCnt: Count of data to write.

bMode: Mode of operation. If mode is set to I2Cm_CompleteXfer, a complete transfer is performed. If mode is set to I2Cm_RepStart, a repeat start is sent instead of start. If mode is set to I2Cm_NoStop, a stop is not sent. This allows an I^{2}C bus combined transfer to be sent to the slave. See the table at the end of this section.

Return Value:

The count of bytes written and acknowledged by the slave is returned. Zero is returned if no bytes were successfully acknowledged by the slave.

Side Effects:

You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, the CUR_PP and IDX_PP page pointer registers are modified.

I2Cm_bWriteCBytes

Description:

Writes one or more bytes (bCnt) to the slave addressed by (bSlaveAddr) from a constant flash array (pbXferData).

C Prototype:

```
BYTE I2Cm_bWriteCBytes(BYTE bSlaveAddr, const BYTE * pbXferData, BYTE bCnt, BYTE
bMode);
```

Assembler:

```
mov   A,I2Cm_CompleteXfer    ; Pass complete transfer flag
push  A
mov   A,0x09                 ; Pass the byte count
push  A
mov   A,>sData               ; Load the MSB of the sData pointer
push  A
mov   A,<sData               ; Load the LSB of the sData pointer
push  A
mov   X,SP                   ; Get the stack location +1
dec   X                      ; X now points to the last byte pushed
```

```
mov   A,0x68                  ; Pass slave address 0x68
lcall  _I2Cm_bWriteCBytes     ; Call function to write data to slave
                              ; A contains return value.
add   sp,-4                   ; Restore the stack
```

Parameters:

bSlaveAddr: 7-bit slave address.

pbXferData: Pointer to “const" data array in flash.

bCnt: Count of data to write.

bMode: Mode of operation. If mode is set to I2Cm_CompleteXfer, a complete transfer is performed. If mode is set to I2Cm_RepStart, a repeat start condition is generated instead of a start condition. If mode is set to I2Cm_NoStop, a stop is not generated. This allows an I^{2}C bus combined transfer to be sent to the slave. See the table at the end of this sub-section.

Note The bMode parameter may be used to perform an I^{2}C bus combined format transfer. To execute a combined transfer, first execute an I2Cm_bWriteBytes or I2Cm_bWriteCBytes command with the bMode parameter set to I2Cm_NoStop (0x02). This performs a write without a stop. Next, execute an I2Cm_fReadBytes command with the bMode parameter set to I2Cm_RepStart (0x01).

Return Value:

The count of bytes written and acknowledged by the slave is returned. Zero is returned if no bytes were successfully acknowledged by the slave. The return value should be compared to the bCnt to verify that all bytes have been transferred. A non-zero return value may be an error condition, if one or more bytes were transferred, and less bytes were transferred than specified by bCnt.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

Table 1. bMode Constants for I2Cm_bWriteBytes, I2Cm_bWriteCBytes, and I2Cm_ fReadBytes

Constant	Value	Description
I2Cm_CompleteXfer	0x00	Perform complete transfer from Start to Stop
I2Cm_RepStart	0x01	Send Repeat Start instead of Start
I2Cm_NoStop	0x02	Execute transfer without a Stop

API Low-Level Functions

For most applications, the low level functions are not required. The low level functions provide greater flexibility for specialized applications.

Table 2. Constants for Low-Level Functions

Constant	Value	Description
I2Cm_WRITE	0x00	Start an I^{2}C write sequence
I2Cm_READ	0x01	Start an I^{2}C read sequence
I2Cm_ACKslave	0x01	ACK slave when reading a byte
I2Cm_NAKslave	0x00	NAK slave when reading a byte

I2Cm_fSendStart

Description:

Generates an I^{2}C bus start condition, sends the address and R/W bit, and then returns the ACK result. The R/W bit is determined by the fRW parameter.

C Prototype:

```
BYTE I2Cm_fSendStart( BYTE bSlaveAddr, BYTE fRW );
```

Assembler:

```
mov   A,0x68                    ; Load slave address
mov   X,I2Cm_WRITE              ; Prepare for a write sequence
lcall  I2Cm_fSendStart           ; Return value in A
```

Parameters:

bSlaveAddr: 7-bit slave address.

fRW: If set to I2Cm_READ, a read sequence is initiated. If set to I2Cm_WRITE, a write sequence is initiated.

Return Value:

If the return value is non-zero, the slave acknowledged the address. If the return value is zero, the slave did not acknowledge the address.

Side Effects:

You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

I2Cm_fSendRepeatStart

Description:

Generates an I^{2}C bus repeat start condition, sends the address and R/W bit, and then returns the ACK result. The R/W bit is determined by the fRW parameter.

C Prototype:

```
BYTE I2Cm_fSendRepeatStart( BYTE bSlaveAddr, BYTE fRW );
```

Assembler:

```
mov   A,0x68                        ; Load address
mov   X,I2Cm_READ                   ; Prepare for a read sequence
```

```
lcall  I2Cm_fSendRepeatStart       ; Return value in A
```

Parameters:

bSlaveAddr: 7-Bit slave address.

fRW: If set to I2Cm_READ, a read sequence is initiated. If set to I2Cm_WRITE, a write sequence is initiated.

Return Value:

If the return value is non-zero, the slave acknowledged the address. If the return value is zero, the slave did not acknowledge the address.

Side Effects:

You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

I2Cm_SendStop

Description:

Generates an I^{2}C bus stop condition.

C Prototype:

```
void I2Cm_SendStop( void );
```

Assembler:

```
lcall I2Cm_SendStop        ; Generate I2C stop condition
```

Parameters:

None

Return Value:

None

Side Effects:

You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

I2Cm_fWrite

Description:

Sends a single byte I^{2}C bus write and ACK. This function does not generate a start or stop condition.

C Prototype:

```
BYTE I2Cm_fWrite( BYTE bData );
```

Assembler:

```
mov   A,[bRamData]          ; Load data to send to slave
lcall  I2Cm_fWrite           ; Initiate I2C write
```

Parameters:

bData: Byte to be sent to slave.

Return Value:

The return value is non-zero if the slave acknowledged the master. The return value is zero if the slave did not acknowledge the master.

Side Effects:

You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

I2Cm_bRead

Description:

Initiates a single byte I^{2}C bus write and ACK phase. This function does not generate a start or stop condition. The fACK flag determines whether the slave is acknowledged upon receiving the data.

C Prototype:

```
BYTE I2Cm_bRead( BYTE fACK );
```

Assembler:

```
mov   A,I2Cm_ACKslave       ; Set flag to ACK slave
lcall  I2Cm_bRead            ; Read single byte from slave
                            ; Return data is in reg A
```

Parameters:

fACK: Set to I2Cm_ACKslave if master should ACK the slave after receiving the data; otherwise, flag should be set to I2Cm_NAKslave.

Return Value:

Byte received from slave.

Side Effects:

You can modify the A and X registers by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

Sample Firmware Source Code

The following are examples in assembly and C for communication with an I^{2}C slave.

```
;-------------------------------------------------------------------------
; Sample assembly code to communicate with the Dallas DS1307 clock/EEROM
; via the I2C interface.
;
; This code sets the time, then reads the 7-byte time string back
; over and over in a loop.
; The address of the DS1307 is 0x68.
;-------------------------------------------------------------------------
include "m8c.inc"       ; part specific constants and macros
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main

area  bss  (RAM)
sRxData:    blk  16
area  text (ROM,REL)

.LITERAL
sTxData:
db    0x00                    ; Slave internal address 0
    db    0x12,0x34,0x08          ; Seconds and minutes in BCD 8:34:12am
    db    0x01                    ; Day of Week, Monday
db    0x15,0x03,0x02          ; Day-Month-year  15-Mar-02
    db    0x93                    ; Enable clock output
 .ENDLITERAL

_main:

    call  I2Cm_Start              ; Initialize I2C master

    mov   A,I2Cm_CompleteXfer      ; Pass normal transfer mode
    push  A
    mov   A,0x09                  ; Pass all 9 bytes of sTxData
    push  A
    mov   A,>sTxData              ; Load the MSB of the sTxData pointer
    push  A
    mov   A,<sTxData              ; Load the LSB of the sTxData pointer
    push  A
    mov   X,SP                    ; Get the stack location +1
    dec   X                       ; That points to the last byte pushed
    mov   A,0x68                  ; Pass slave address 0x68
    call  _I2Cm_bWriteCBytes      ; Call function to write data to slave
                                  ; A contains return value.
    add   SP,-4                   ; Restore the stack

RxLoop:                           ; Keep reading the time from the
                              ; Dallas DS1307

    ; Do a combined transfer, write then read
; The write sets the sub-address value to 0x00.  The read then
    ; starts reading the values starting at the sub-address 0x00.
    ;
    mov   A,I2Cm_NoStop           ; Don't generate a stop sequence
    push  A
    mov   A,01h                   ; Write only the first byte of the string
    push  A                       ; which is the internal sub-address.
    mov   A,>sTxData              ; Load the MSB of the sTxData pointer
    push  A
    mov   A,<sTxData              ; Load the LSB of the sTxData pointer
    push  A
    mov   X,SP                    ; Get the stack location +1
    dec   X                       ; Dec the pointer to point to the last byte
    mov   A,0x68                  ; Pass slave address 0x68
    call  _I2Cm_bWriteCBytes      ; Call function to write data to slave
```

```
                                 ; Reg A contains return value.
    add   SP,-4                  ; Restore the stack

mov   A,I2Cm_RepStart        ; Start with a Repeat start
    push  A
    mov   A,0x07                 ; Read just the 7 time bytes back
    push  A
    mov   A,>sRxData             ; Load the MSB of the sRxData pointer
    push  A
    mov   A,<sRxData             ; Load the LSB of the sRsData pointer
    push  A
    mov   X,SP                   ; Get the stack location +1
    dec   X                      ; That points to the last byte pushed
    mov   A,0x68                 ; Pass slave address 0x68
    call  _I2Cm_fReadBytes       ; Call function to read data from slave
                                 ; A contains return value.
    add   SP,-4                  ; Restore the stack

    jmp   RxLoop                 ; Setting a breakpoint here, you should
                                 ; be able to see the returned time data
                                 ; in the sRxData RAM locations.
    ret
```

Sample code in C is as follows.

```
//-------------------------------------------------------------------------
// Sample C code to communicate with the Dallas DS1307 clock/EEROM
// via the I2C interface.
//
// This code sets the time, then reads the 7-byte time string back
// over and over in a loop.
// The address of the DS1307 is 0x68.
//-------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

BYTE rxBuf[8];

const BYTE txCBuf[] = { 0x00,          // Slave internal sub-address 0
                        0x12,0x34,0x08,  // Seconds and minutes in BCD
                                         // 8:34:12am
                        0x01,          // Day of Week, Monday
                    0x15,0x03,0x02,  // Day-Month-year  15-Mar-02
                        0x93 };          // Enable clock output

void main(void)
{
    BYTE status;                         // I2C communication status
    BYTE i;                              // Temp counter variable

I2Cm_Start();                        // Initialize I2C Master interface

    // Set the time
    status = I2Cm_bWriteCBytes(0x68,txCBuf,9,I2Cm_CompleteXfer);
    // In a endless loop, keep reading the time from the DS1307
    do {

       // Write sub-address to DS1307
       // Perform a combined transfer by leaving off the Stop on the Write
       // command and beginning the Read with a Repeat Start.

       I2Cm_bWriteCBytes(0x68,txCBuf,1,I2Cm_NoStop );

   status = I2Cm_fReadBytes(0x68,rxBuf,7,I2Cm_RepStart );

   if(status == 0) {
       // Flag an error condition
       }

       // This next section of code performs exactly the same sequence
       // as the above code, but with low level commands.

       I2Cm_fSendStart(0x68,I2Cm_WRITE);        // Do a write
       I2Cm_fWrite(0x00);                       // Set sub address
                                                // to zero
       I2Cm_fSendRepeatStart(0x68,I2Cm_READ);   // Do a read

       for(i = 0; i < 6; i++) {
          rxBuf[i] = I2Cm_bRead(I2Cm_ACKslave); // Read first 6 bytes,
                                                // and ACK the slave
       }

       rxBuf[7] = I2Cm_bRead(I2Cm_NAKslave);    // Read data byte and
                                                // NAK the slave to signify
                                                // end of read.
       I2Cm_SendStop();

    } while(1);

}
```

Version History

Version	Originator	Description
1.4	DHA	Added Version History
2.00	MYKZ	Added DM2 register initialization into Start() function to eliminate communication issues.

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.