Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

I²C Hardware Block Datasheet I2CHW V 2.00

Copyright © 2003-2015 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/24/22/21xxx, CY8C23x33, CY7C603xx, CY7C64215, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx, CY8C21x12						
Slave	0	0	0	374 - 591	6 - 11	2
Master	0	0	0	1031 - 1085	7 - 11	2
Multi Master Slave	0	0	0	1331 - 1944	14 - 22	2

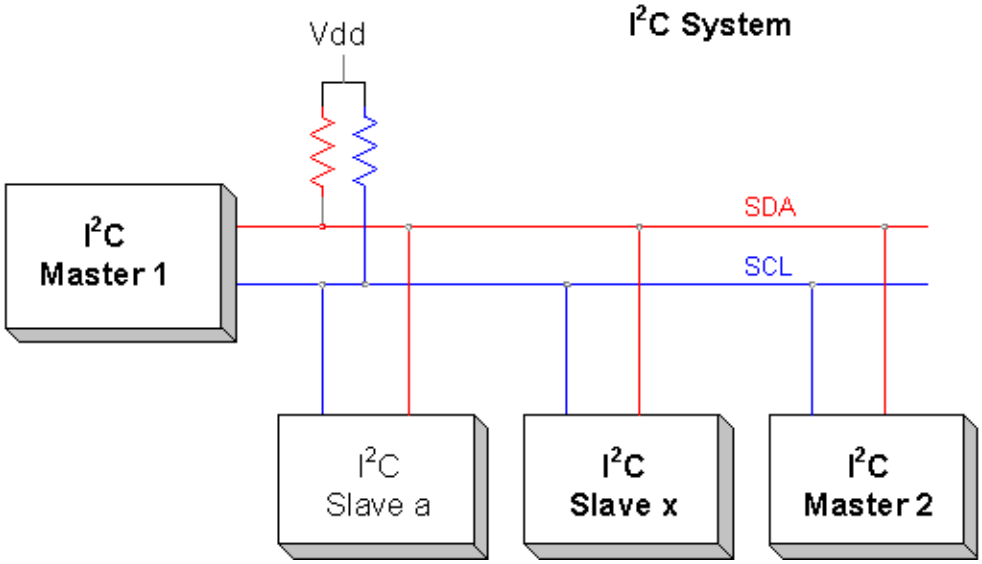
For one or more fully configured, functional example projects that use this user module go to www.cypress.com/psocexampleprojects.

Features and Overview

- Industry standard Philips I²C bus compatible interface
- Master and Slave operation, Multi Master capable
- Only two pins (SDA and SCL) required to interface to I²C bus
- Standard data rate of 100/400 kbps, also supports 50 kbps
- High level API requires minimal user programming
- 7-bit addressing mode

The I²C Hardware User Module implements an I²C device in firmware. The I²C bus is an industry standard, two-wire hardware interface developed by Philips®. The master initiates all communication on the I²C bus and supplies the clock for all slave devices. The I2CHW User Module supports the standard mode with speeds up to 400 kbps. No digital or analog user blocks are consumed with this module. The I2CHW User Module is compatible with other slave devices on the same bus.

Figure 1. I²C Block Diagram



Functional Description

This user module gives support for an I²C hardware resource. It is capable of transferring data at 50/100/400 kbps when the CPU clock is configured to run at 12 MHz. Faster or slower CPU clocks may be used, but may result in more or less bus stalling during address or data processing. The I²C specification allows the master to run at clock speeds from 100 kHz down to DC. There are two different selections for SDA and SCL providing direct access to the hardware resource. Seven-bit address mode is supported in the supplied APIs. This module does not require any analog or digital PSoC blocks to transfer data.

The I²C resource supports data transfer at a byte-by-byte level. At the end of each address or data transmission/reception, status is reported or a dedicated interrupt may be triggered. Status reporting and interrupt generation is dependent on the direction of data transfer and the condition of the I²C bus as detected by the hardware. Interrupts may be configured to occur on byte-complete, bus-error detection and arbitration loss.

Every I²C transaction consists of a Start, Address, R/W Direction, Data, and a Termination. The I²C resource used for this user module is capable of operating as either an I²C Master or I²C Slave. For either the Master or Slave operation, this user module gives an interrupt based buffered transfer mechanism. Communication is initiated from a foreground function call. At the completion of each byte of the message, an interrupt is triggered and the I²C bus is stalled, the interrupt service routine (ISR) given takes appropriate action on the bus allowing communication to continue, depending on the initialization performed by the user. A slave device which does not acknowledge an address is not interrupted again until the next address is received. Slave devices must respond to each address either by acknowledging or not-acknowledging.

Ignoring differences between a Master and Slave, two general cases exist, that of a receiver and that of a transmitter. For an I²C receiver, an interrupt occurs after the 8th bit of incoming data. At this point a receiving device must decide to acknowledge (ack) or not-acknowledge (nak) the incoming byte whether it is an address or data. The receiving device then writes appropriate control bits to the I2C_SCR register, informing the I²C resource of the ack/nak status. The write to the I2C_SCR register paces data flow on the bus by uninstalling the bus, placing the ack/nak status on the bus and shifting the next data byte in. For the

second case of a transmitter, an interrupt occurs after an external receiving device has given an ack or nak. The I2C_SCR may be read to determine the status of this bit.

For a transmitter, data would be loaded into the I2C_DR register and the I2C_SCR register is again written to trigger the next portion of the transmission.

When using the buffered read and write routines (bWriteBytes(...), bWriteCBytes(...), fReadBytes(...)), it is not necessary to use any of the buffer initialization functions (InitWrite(...), InitRamRead(...), InitFlashRead(...)). These functions are called as part of the function call that initiates the buffered read or write (bWriteBytes(...), bWriteCBytes(...), fReadBytes(...)).

In addition to the buffer based transactions supported, the I^2C master User Module supports polled data transfers on a byte by byte basis without using the interrupt or supplied ISR. All APIs are described in this document.

Detailed descriptions of the I^2C bus and the implementation of the resource here may be obtained by referring to the complete I^2C specification available on the internet, and by referring to the device datasheet supplied with PSoC Designer.

I2C and Sleep

Special care must be taken when using I2C with a project that goes to a sleep state. Before the project enters a sleep state, follow these steps need for proper sleep entry, and proper I2C handling:

1. Ensure that all I2C traffic is complete.
2. Disable the I2C by calling the Stop API.
3. Configure the I2C pins to a analog High-Z drive mode.

Follow these steps when the part wakes from sleep:

1. Ensure that there is no active I2C traffic.
2. Enable the I2C by calling the Start API.
3. Configure the I2C pins to Open-Drain Drives Low drive mode.
4. Enable Interrupts.

MultiMasterSlave Operation

MultiMasterSlave operation is an extension of both the Master and Slave and combines the two. As such the implementation also uses the most code to implement the APIs and ISR. The features that the MultiMasterSlave adds are:

1. Multiple Masters may reside on the same bus. If more than one Master attempts to talk on the bus, a mechanism exists so that they do not conflict. This mechanism is called arbitration. It is completely discussed in the Philips I^2C specification.

2. Each Master may also operate as a slave with an associated slave address. Another Master may address the ‘slave’ instance of any other master as if it were a slave. If a Master loses arbitration on an address which is its own slave address it correctly responds as a slave for the duration of the I^2C transaction.

MultiMasterSlave APIs

Function calls for the MultiMasterSlave implementation are nearly identical to those used for the Master and Slave. However, because the MultiMasterSlave implements a master and slave device, the names of the function calls have been modified with the appropriate designation. For example: where the Slave (and Master) use a function “InitWrite(...)”, the MultiMasterSlave gives two functions designated: “InitMasterWrite(...)” and InitSlaveWrite(...)” The initialization functions should be used as they would for the appropriate single Master or single Slave device. To further expand on this topic, when the Master portion of the MultiMasterSlave is used for a buffered read/write, the appropriate Init function is called internally from the bWrite() or fRead() function (and would not therefore be normally used directly by a user). On the other hand, the InitSlave set of functions WOULD be used by a programmer to set up an area of memory for a later access by an external master.:

Master operation and Slave operation of the MultiMasterSlave User Module are enabled separately. In other words, both the EnableMaster() and EnableSlave() functions must be called to enable Slave and Master functionality.

Design Considerations

- Unlike the Software implementation of I^2C, this implementation runs using internally generated clocks. The main oscillator may be set to any clock speed. Data throughput may be affected by processor speed but byte-by-byte data transfer functions at the specified I^2C speed.
- Slave devices maintain an internal count of the buffer space remaining for access by a Master. For the MultiMasterSlave, the count variables are called UMNAME_SlaveWrite_Count and UMNAME_SlaveRead_Count. The Slave(only) the variables are named UMNAME_Read_Count and UMNAME_Write_Count. The variables are global and may be accessed from a users C or assembly code by including an appropriate 'extern' declaration. A user may determine the number of bytes read or written by a master by subtracting the current value of the count variable from the initial value of the count variable. The initial size of the count variable is set by using the functions UMNAME_InitWrite, UMNAME_InitRamRead, UMNAME_InitFlashRead in the case of the Slave (only) user module. The function calls setting the count value for the optional slave used in the MultiMasterSlave User Module are UMNAME_InitSlaveWrite, UMNAME_InitSlaveRamRead, and UMNAME_InitSlaveFlashRead.
- Reading and writing data within the I^2C slave is accomplished using buffers. The user must initialize appropriate buffers before the I^2C slave is enabled. After a read or a write has been initiated by the I^2C master, appropriate status bits are set in the I2CHW_Status byte. The foreground process in the slave can then operate on data deposited in a write buffer or extracted from a read buffer. The slave data transfer routine (ISR) does not allow buffers to be accessed beyond their defined length when the ISR is entered. Reading and writing to buffers is handled as follows.

1. If the I^2C master attempts to read more data than is contained in a buffer, the last byte is retransmitted until the I^2C master stops reading. (The I^2C protocol does not define a method for the I^2C slave to stop a master from reading.)
2. When an I^2C master is writing one or more data bytes to the I^2C slave, upon receiving the last byte for which storage is available, the slave generates a NAK. If the I^2C master continues to write data, the slave continues to NAK it. After the first NAK is generated (data is stored in the last available location), further data is not stored.
3. For Slave read and write buffer transactions, an ‘error’ condition is set when the last byte of the buffer is used. The reasoning behind this is that since the slave has no control over how many bytes a master device reads or writes, once the last byte of a buffer is used there is no way to be assured that some sort of over-run did NOT occur. For this reason, to avoid overflow conditions using (slave) read buffers,

and NAKs to the master during (slave) write operations, read and write buffers should be set to be one byte longer than the maximum expected to be used by the master. For example, a one byte slave write buffer always causes the slave to NAK the master when one byte is written. Writing one byte to a two byte buffer results in an ACK to the master.

4. If a buffer is defined with zero length, data written to the I^2C slave is NAK'ed and is not stored.

Enabling the ability to read data directly from flash also allows the use of either RAM or flash buffers. Whether the data transfer ISR uses a read buffer located in flash/ROM or RAM, it can be configured using supplied APIs.

Dynamic Reconfiguration

Incorporating the I2CHW resource into Dynamically loaded/unloaded overlays is not recommended. The I2CHW resource should be placed as part of the base configuration only. Operation of the I2CHW block may be modified as operational requirements dictate, but attempting to 'remove' the resource as part of dynamic reconfiguration may result in adverse effects on external I^2C devices.

I^2C Addressing

I^2C addresses are contained in the upper 7-bits of the first byte of a read or write transaction. This byte is used by the I^2C master to address the slave. Valid selections are from 0-127 decimal. The LSB of the byte contains the R/~W bit. If this bit is 0, the address is written to. If the LSB is a 1, then the addressed slave has data read from it.

Internally, the user module takes the input address, shifts it and combines it with a read/write bit to construct a complete address byte.

Example

An address of 0x48 is passed as a parameter or defined as a slave address. 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.

Since the slave module accepts decimal based numerical input for its address parameter, the 7-bit address must also be entered in decimal (decimal 72).

DC and AC Electrical Characteristics

As the block diagram illustrates, the I^2C bus requires external pull-up resistors. 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 not be 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 1.5 kΩ. The maximum value for R_P is dependent on the bus capacitance and the clock speed. For a 5V system with a bus capacitance of 150 pF, the pull-up resistors should be no larger than 6 kΩ. For more information on "The I^2C-Bus Specification", see the Philips web site at www.philips.com.

One common design consideration with I2C is the size of the pull-up resistors. For most designs the pull-up resistors are between 1.5k and 6k. The selected size of the resistors depends on the communication frequency and the bus capacitance. A greater bus capacitance and pull-up resistor size causes greater rise time on the clock and data lines. The I2C specification provides a maximum rise time. If the rise time on the bus exceeds this maximum, I2C communication will not occur properly. The pull-up resistors must

be sized properly to prevent large rise times. The I2C specification offers the graph to determine the size of the pull-up resistors. For more information please refer to the I2C specification.

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 system, given that the system conforms to the I^2C Standard Specification as defined by Philips.

Placement

The I2CHW User Module allows two choices of SCL and SDA P1[5]/P1[7] or P1[0]/P1[1] and does not require any digital or analog PSoC blocks. There are no placement restrictions. Placement of multiple I^2C modules is not possible since the I^2C module uses a dedicated PSoC resource block and interrupt.

Parameters and Resources

All buffer names are written to describe their use by an I^2C master. For example, the I2Cs_pRead_Buf refers to the location in RAM containing data to be read by the I^2C master.

Slave_Addr

This is a Slave and MultiMasterSlave parameter. It selects the 7-bit slave address which is used by the I^2C master to address the slave or the MultiMasterSlave when it is in slave mode. Valid selections are from 0-127(dec).

Auto_Addr_Check

Selects whether the hardware address feature is disable or enable. If set to disable then the hardware address comparison feature is not available. Setting this option to enable allows I2C block does not support the special system address definition.

This parameter is user configurable only for CY8C28045 device.

I2C_Clock

Specifies the desired clock speed at which to run the I^2C interface. There are three possible clock rates available:

- 50K Standard
- 100K Standard
- 400K Fast

Note The I2C clock is based off a SysClk of 24 MHz. If SysClk is less than 24 MHz the I2C clocks will scale down. For example, if SysClk is 6 MHz the possible clock speeds are 12.5K, 25K, and 100K. SysClk is separate from the CPU clock.

I2C_Pin

Selects the pins from Port 1 to be used for I^2C signals. There is no need to select the proper drive mode for these pins, PSoC Designer does this automatically.

Note Changing the default drive mode prevents the pin from being driven in the proper manner and causes undesired operation on the bus.

Read_Buffer_Types

Selects what types of buffers are supported for data reads. Two selections are available: RAM ONLY or RAM OR FLASH. Selection of RAM ONLY removes code and variables required to support direct

flash-ROM reads. Selection of RAM OR FLASH gives code and variable support for reading either RAM buffers or flash-ROM buffers for data to be transmitted to the master. If RAM OR FLASH is selected, API calls may be used to select whether a RAM or flash-read buffer is used.

Communication_Service_Type

This parameter allows the user to select between an Interrupt based data processing strategy or a polled strategy. In the Interrupt based strategy, a transfer is initiated against a predefined buffer. Data is then moved in or out of the buffer as quickly as possible in the background. An ISR routine is included which handles data movement. When the Polled data processing strategy is selected, the user is in control of when data movement takes place. To implement a polled strategy the user must periodically call the function I2CHW_Poll() (see the I2C.h files for the exact instance name). Each time the polling function is called a single byte is transferred. Other I^2C functions are used identically. The Polled communication strategy may be used in a situation where interrupt latency is critically important (and asynchronous communication interrupts may cause problems). Another use might be when the user desires absolute control of when data is transferred. A drawback of polling is that when the I^2C state machine is enabled, the bus is stalled automatically after each byte until the polling function is called. The polling function is available only for the Slave and MultiMasterSlave implementations of I^2C. The Single Master implementation offers API functions to support byte-wise data transfers.

Interrupt Generation Control

There is an additional parameter that becomes available when the **Enable interrupt generation control** check box in PSoC Designer is checked. This is available under **Project > Settings > Chip Editor**. Interrupt Generation Control is important when multiple overlays are used with interrupts shared by multiple user modules across overlays:

IntDispatchMode

The IntDispatchMode parameter is used to specify how an interrupt request is handled for interrupts shared by multiple user modules existing in the same block but in different overlays. Selecting **ActiveStatus** causes firmware to test which overlay is active before servicing the shared interrupt request. This test occurs every time the shared interrupt is requested. This adds latency and also produces a nondeterministic procedure of servicing shared interrupt requests, but does not require any RAM. Selecting **OffsetPreCalc** causes firmware to calculate the source of a shared interrupt request only when an overlay is initially loaded. This calculation decreases interrupt latency and produces a deterministic procedure for servicing shared interrupt requests, but at the expense of a byte of RAM.

Application Programming Interface

The Application Programming Interface (API) firmware gives high level commands that support sending and receiving multibyte transfers. Read buffers may be set up in RAM or flash memory. Write buffers may only be set up in RAM memory.

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 before 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 may do so in the future.

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.

Common Functions

The following functions are common to the Master, Slave, and the MultiMasterSlave versions of the I2CHW User Module:

I2CHW_Start

Description:

Does Nothing. Given for interface consistency only.

C Prototype:

```
void  I2CHW_Start(void);
```

Assembler:

```
lcall  I2CHW_Start
```

Parameters:

None

Return Value:

None

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.

I2CHW_Stop

Description:

Disables the I^2CHW by disabling the I^2C interrupt.

C Prototype:

```
void   I2CHW_Stop(void);
```

Assembler:

```
lcall  I2CHW_Stop
```

Parameters:

None

Return Value:

None

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.

I2CHW_EnableInt

Description:

Enables I^2C interrupt allowing start condition detection. Remember to call the global interrupt enable function by using the macro: M8C_EnableGInt.

C Prototype:

```
void   I2CHW_EnableInt(void);
```

Assembler:

```
lcall I2CHW_EnableInt
```

Parameters:

None

Return Value:

None

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.

I2CHW_DisableInt

Description:

Disables I^2C's slave by disabling the SDA interrupt. Performs the same action as I2Cs_Stop.

C Prototype:

```
void   I2CHW_DisableInt(void);
```

Assembler:

```
lcall I2CHW_DisableInt
```

Parameters:

None

Return Value:

None

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.

I2CHW_Poll (available in Slave and MultiMasterSlave)

Description:

Used when the Communication_ Service_Type parameter is set to “Polled”. This function gives a user controlled entry into the I/O processing routine. If Communication_Service_Type parameter is set to “Interrupt”, the function does nothing.

Note: Calling I2CHW_Poll releases the I2C bus. This allows a master to possibly read or write data before slave firmware has finished processing the previous request.

C Prototype:

```
void   I2CHW_Poll(void);
```

Assembler:

```
lcall I2CHW_Poll
```

Parameters:

None

Return Value:

None

Side Effects:

One I^2C event is processed each time this routine is called and status variables are updated. An event constitutes either an error condition, an I/O byte, or, in certain cases, a stop condition. There are three possible results from calling this routine:

1. No action if no data was available.
2. Reception or transmission of an address or data byte if one was available.
3. Processing of a stop event when an external master has completed its write operation. When a stop state is processed at the end of a write operation, only status variables are updated. If an I^2C byte is pending when a stop state is processed, the I2CHW_Poll function must be called again to process it.

The I2CHW_Poll() function has no effect if Communication_Service_Type is set to Interrupt. When a start/restart condition and an address is detected on the bus, the bus is stalled until the I2CHW_Poll() function is called. If the address is NAK'ed, subsequent bytes transferred for that transaction are ignored until another start/restart and address is detected, otherwise the I^2C bus is stalled for each data byte until the I2CHW_Poll() function is called. The I^2C data is stalled by the I^2C hardware until this function is called if the Communication_Service_Type is set to Polled. For received data the bus is stalled at the end of the byte and before an ACK/NAK is generated by holding the SCL (clock) line low. For transmitted data the bus is stalled immediately after the ACK/NAK bit is generated externally.

I2CHW_ResumeInt

Description:

Re-enables I^2C interrupt allowing start condition detection. This API enables the I^2C interrupt in the INT_MSK3 register and does not clear the I^2C interrupt in the INT_CLR3 register. Remember to call the global interrupt enable function by using the macro: M8C_EnableGInt.

C Prototype:

```
void I2CHW_ResumeInt(void);
```

Assembler:

```
lcall I2CHW_ResumeInt
```

Parameters:

None

Return Value:

None

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.

I2CHW_ClearInt

Description:

Clears the I^2C interrupt in the INT_CLR3 register. Remember to call the global interrupt enable function by using the macro: M8C_EnableGInt.

C Prototype:

```
void I2CHW_ClearInt(void);
```

Assembler:

```
lcall I2CHW_ ClearInt
```

Parameters:

None

Return Value:

None

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.

API Low Level Communication Functions (available in Master and MultiMasterSlave)

For most applications, the low level functions are not required. The low level functions give greater flexibility for specialized applications. They do not use the ISR. In most cases if the I2C interrupt is enabled a call to one of the low level routines explicitly disables it.

I2CHW_fSendStart

Description:

Generates an I^2C 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 I2CHW_fSendStart( BYTE bSlaveAddr, BYTE fRW );
```

Assembler:

```
mov   A,0x68                  ; Load slave address
mov   X,I2CHW_WRITE           ; Prepare for a write sequence
lcall  I2CHW_fSendStart        ; Return value in A
```

Parameters:

bSlaveAddr: 7-bit slave address. fRW: If set to I2CHW_READ, a read sequence is initiated. If set to I2CHW_WRITE, a write sequence is initiated.

Return Value:

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

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.

The I2CHW interrupt is disabled if previously enabled.

Constant	Value	Description
I2CHW_WRITE	0x00	Start an I^2C write sequence
I2CHW_READ	0x01	Start an I^2C read sequence
I2CHW_ACKslave	0x01	ACK slave when reading a byte
I2CHW_NAKslave	0x00	NAK slave when writing a byte

I2CHW_fSendRepeatStart

Description:

Generates an I^2C 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 I2CHW_fSendRepeatStart( BYTE bSlaveAddr, BYTE fRW );
```

Assembler:

```
mov   A,0x68                    ; Load address
mov   X,I2CHW_READ               ; Prepare for a read sequence
lcall  I2CHW_fSendRepeatStart     ; Return value in A
```

Parameters:

bSlaveAddr: 7-Bit slave address. fRW: If set to I2CHW_READ, a read sequence is initiated. If set to I2CHW_WRITE, a write sequence is initiated.

Return Value:

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

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. The I2CHW interrupt is disabled if previously enabled.

Sets the RepStart flag in the bStatus byte. This prevents the polled the routine from hanging when used with polling interface. Care should be taken mixing buffered commands with low level

commands. With the polled interface option in particular if this routine is entered without a ‘byte complete’ flag set in the I2C_SCR register, it does not exit.

I2CHW_SendStop

Description:

Generates an I^2C bus stop condition.

C Prototype:

```
void I2CHW_SendStop( void );
```

Assembler:

```
lcall I2CHW_SendStop        ; Generate I2C stop condition
```

Parameters:

None

Return Value:

None

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.

I2CHW_fWrite

Description:

Sends a single-byte I^2C bus write and ACK. This function does not generate a start or stop condition. This routine should ONLY be called when a previous start and address has been generated on the I2C bus. It should only be used when I2C_BYTE_COMPL is set in the I2C_SCR register.

C Prototype:

```
BYTE I2CHW_fWrite( BYTE bData );
```

Assembler:

```
mov   A,[bRamData]             ; Load data to send to slave
lcall  I2CHW_fWrite             ; Initiate I2C write
```

Parameters:

bData: Byte to be sent to slave.

Return Value:

The return value is nonzero, if the slave acknowledged the master. The return value is zero, if the slave did not acknowledge the master. If the Slave failed to acknowledged the Master, the value of bStatus is 0xff.

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. The I2CHW interrupt is disabled if previously enabled.

I2CHW_bRead

Description:

Initiates a single-byte I^2C bus read 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. This routine should ONLY be called when a previous start and address has been generated on the I2C bus. It should only be used when I2C_BYTE_COMPL is set in the I2C_SCR register. If fACK is set, it should be followed by next I2CHW_bRead call. To finish the read transaction Master should call this function with I2CHW_NAKslave parameter.

C Prototype:

```
BYTE I2CHW_bRead( BYTE fACK );
```

Assembler:

```
mov   A,I2CHW_ACKslave        ; Set flag to ACK slave
lcall  I2CHW_bRead             ; Read single byte from slave
                              ; Return data is in reg A
```

Parameters:

fACK: Set to I2CHW_ACKslave if master should ACK the slave after receiving the data; otherwise, flag should be set to I2CHW_NAKslave. In general, the ACK from master means request for the next data byte from the Slave. If set to I2CHW_ACKslave, the master after receiving current data byte and ACKing will immediately clock in the next byte from the slave. This next byte will be returned the next time I2CHW_bRead() is called. If set to I2CHW_NAKslave, the master will only NAK the current byte and will not clock in a the next byte of data.

Return Value:

Byte received from slave.

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. The I2CHW interrupt is disabled if previously enabled.

Slave Functions

The following functions are specific to the Slave version of the I2CHW User Module.

I2CHW_EnableSlave

Description:

Enable the I^2C Slave function for the I^2C HW block by setting the Enable Slave bit in the I2C_CFG register.

C Prototype:

```
void   I2CHW_EnableSlave(void);
```

Assembler:

```
lcall  I2CHW_EnableSlave
```

Parameters:

None

Return Value:

None

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. As noted in the CY8C27xxxA datasheet, the I^2C configuration registers should be written at 6 MHz CPU clock speed. The routine implemented here does that. If writing directly to the configuration registers, care should be taken to correctly perform the operation or I^2C communication failures may occur.

I2CHW_DisableSlave

Description:

Disables the I^2C Slave function by clearing the Enable Slave bit in the I2C_CFG register.

C Prototype:

```
void   I2CHW_DisableSlave(void);
```

Assembler:

```
lcall  I2CHW_DisableSlave
```

Parameters:

None

Return Value:

None

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. As noted in the CY8C27xxxA datasheet, the I^2C configuration registers should be written at 6 MHz CPU clock speed. The routine implemented here does that. If writing directly to the configuration registers, care should be taken to correctly perform the operation or I^2C communication failures may occur.

I2CHW_bReadI2CStatus

Description:

Returns the status bits in the Control/Status register.

C Prototype:

```
BYTE  I2CHW_bReadI2CStatus(void);
```

Assembler:

```
lcall  I2CHW_bReadI2CStatus ; Accumulator contains status
```

Parameters:

None

Return Value:

```
BYTE  I2CHW_RsrcStatus
```

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.

Constant	Value	Description
I2CHW_RD_NOERR	01h	Data read by the master, normal ISR exit
I2CHW_RD_OVERFLOW	02h	More data bytes were read by the master than were available
I2CHW_RD_COMPLETE	04h	A read was initiated and is complete
I2CHW_READFLASH	08h	The next read is from a flash location
I2CHW_WR_NOERR	10h	Data was written successfully by the master
I2CHW_WR_OVERFLOW	20h	The master wrote too many bytes for the write buffer
I2CHW_WR_COMPLETE	40h	A master write was completed by a new address or stop
I2CHW_ISR_ACTIVE	80h	The I^2C ISR has not yet exited and is active

I2CHW_ClrRdStatus

Description:

Clears the read status bits in the I2CHW_RsrcStatus register. No other bits are affected.

C Prototype:

```
void  I2CHW_ClrRdStatus (void);
```

Assembler:

```
lcall  I2CHW_ClrRdStatus
```

Parameters:

None

Return Value:

None

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.

I2CHW_ClrWrStatus

Description:

Clears the write status bits in the I2CHW_RsrcStatus register. No other bits are affected.

C Prototype:

```
void  I2CHW_ClrWrStatus (void);
```

Assembler:

```
lcall  I2CHW_ClrWrStatus
```

Parameters:

None

Return Value:

None

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.

I2CHW_InitWrite

Description:

Initializes a data buffer pointer for the slave to use to deposit data, and initializes the value of a count byte for the same buffer. Count is initialized to the maximum supplied buffer length. On the next instance of a master write, data is placed at the address defined by this function.

C Prototype:

```
void   I2CHW_InitWrite(BYTE * pWriteBuf, BYTE bBufLen);
```

Assembler:

```
AREA    bss (RAM, REL)
abWriteBuf    blk 10h

AREA    text (ROM,REL)
    push X                      ; save registers
    push A
    add  SP, 3
    mov  X, SP
    dec  X                      ; X points at data SP points at next
                                ; empty stack location
    mov  [X], <abWriteBuf       ; place the buffer address
    mov  [X-1], >abWriteBuf     ; (page 0)on the stack at [X]
    mov  [X-2], 10              ; place the count at [x-2]
                                ; don't care what [X-1] is
                                ; the compiler would assign 0 as the
                                ; MSB of the Ramtbl addr
    lcall  I2CHW_InitWrite
    add   SP, -3                ; restore the stack
    pop   A                     ; restore registers
```

```
    pop   X
```

Parameters:

pWriteBuf: Pointer to a RAM buffer location. buf_len: Length of write buffer.

Return Value:

None

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.

I2CHW_InitRamRead

Description:

Initializes a flash data buffer pointer from which the slave retrieves data, and initializes the value of a count byte for the same buffer. Clears the I2CHW_SlaveStatus flag I2CHW_READFLASH to 0, causing the next read to be attempted from a *previously* set buffer location in RAM.

C Prototype:

```
void   I2CHW_InitRamRead(BYTE * pReadBuf, BYTE bBufLen);
```

Assembler:

```
AREA bss (RAM,REL)
abReadBuf:    blk 10h

AREA text (ROM,REL)
    push  X                       ; save registers
    push  A
add   SP, 3
    mov   X, SP
    dec   X                       ; X points at data SP points at next
                                  ; empty stack location
    mov   [X], <abReadBuf         ; place the read buffer address
    mov   [X-1], >abReadBuf       ; (page0)on the stack at [X]
    mov   [X-2], 10               ; place the count at [x-2]
                                  ; don't care what [X-1] is
                                  ; the compiler would assign 0 as
                                  ; the MSB of the Ramtbl addr
    lcall  I2CHW_InitRamRead
    add   SP, -3                  ; back up the stack (subtract 3)
    pop   A                       ; restore registers
    pop   X
```

Parameters:

_ReadBuf: Pointer to a RAM buffer location. bBufLen: Length of read buffer.

Return Value:

None

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.

I2CHW_InitFlashRead

Description:

Initializes a flash data buffer pointer for retrieval of data. Sets the I2CHW_SlaveStatus flag I2CHW_READFLASH to 1, causing the next read to be attempted from a *previously* set buffer location in flash.

C Prototype:

```
void   I2CHW_InitFlashRead(const BYTE * pFlashBuf, WORD wBufLen);
```

Assembler:

```
area table(ROM,ABS)
org 0x1015

abFlashBuf:

      db 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88
      db 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff


area text(ROM,REL)


    push  X                         ; save registers
    push  A
    add   SP, 4
    mov   X, SP
    dec   X                         ; X points at data SP points at next
                      ; empty stack location
    mov   [X], <abFlashBuf          ; place the LSB of rom
                      ; address on the stack at [X]
    mov   [X-1], >abFlashBuf        ; place the MSB of the rom address
                      ; at [x-1] variable
    mov   [X-2], 0x0F               ; place the LSB of length
                                    ; at [x-2]
    mov   [X-3], 0x00               ; place the MSB of length
                      ; at [x-3]
    lcall  I2CHW_InitFlashRead
    add   SP, -4                    ; adjust the stack (subtract 4)
    pop   A                         ; restore registers
    pop   X
```

Parameters:

pFlashBuf: Pointer to a flash/ROM buffer location. wBufLen: Length of buffer.

Return Value:

None

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.

Master Functions

The following functions are specific to the Master version of the I2CHW User Module.

Note Two different methods of passing API parameters are supported within the given 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 users 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 in this section (new style parameter passing) with assembly language implementations by adding an underscore to the assembly call statement. No applications written in C are affected. This note applies only to Master version of the I2CHW User Module. Routines that fit this description are: I2CHW_fReadBytes, I2CHW_bWriteBytes, and I2CHW_bWriteCBytes.

I2CHW_EnableMstr

Description:

Enables the I^2C HW block as a Master by setting the Enable Master bit in the I2C_CFG register.

C Prototype:

```
void   I2CHW_EnableMstr(void);
```

Assembler:

```
lcall I2CHW_EnableMstr
```

Parameters:

None

Return Value:

None

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.

As noted in the CY8C27xxxA datasheet, the I^2C configuration registers should be written at 6 MHz CPU clock speed. The routine implemented here does that. If writing directly to the configuration registers, care should be taken to correctly perform the operation or I^2C communication failures may occur.

I2CHW_DisableMstr

Description:

Disables the I^2C Master function by clearing the Enable Master bit in the I2C_CFG register.

C Prototype:

```
void   I2CHW_DisableMstr(void);
```

Assembler:

```
lcall  I2CHW_DisableMstr
```

Parameters:

None

Return Value:

None

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. As noted in the CY8C27xxxA datasheet, the I^2C configuration registers should be written at 6 MHz CPU clock speed. The routine implemented here does that. If writing directly to the configuration registers, care should be taken to correctly perform the operation or I^2C communication failures may occur.

I2CHW_bReadI2CStatus

Description:

Returns the status bits in the Control/Status register.

C Prototype:

```
BYTE  I2CHW_bReadI2CStatus(void);
```

Assembler:

```
lcall  I2CHW_bReadI2CStatus ; Accumulator contains status
```

Parameters:

None

Return Value:

```
BYTE  I2CHW_RsrcStatus
```

See Table.

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.

Constant	Value	Description
I2CHW_RD_NOERR	01h	Data read by the master, normal ISR exit
I2CHW_RD_ OVERFLOW	02h	More data bytes were read by the master than were available
I2CHW_RD_ COMPLETE	04h	A read was initiated but has not yet been completed
I2CHW_READFLASH	08h	The next read is from a flash location
I2CHW_WR_NOERR	10h	Data was written successfully by the master
I2CHW_WR_OVERFLOW	20h	The master wrote too many bytes for the write buffer
I2CHW_WR_COMPLETE	40h	A master write was completed by a new address or stop
I2CHW_ISR_ACTIVE	80h	The I^2C ISR has not yet exited and is active

I2CHW_ClrRdStatus

Description:

Clears the read status bits in the I2CHW_RsrcStatus register. No other bits are affected.

C Prototype:

```
void  I2CHW_ClrRdStatus (void);
```

Assembler:

```
lcall  I2CHW_ClrRdStatus
```

Parameters:

None

Return Value:

None

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.

I2CHW_ClrWrStatus

Description:

Clears the write status bits in the I2CHW_RsrcStatus register. No other bits are affected.

C Prototype:

```
void  I2CHW_ClrWrStatus (void);
```

Assembler:

```
lcall  I2CHW_ClrWrStatus
```

Parameters:

None

Return Value:

None

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.

I2CHW_fReadBytes

Description:

This is a Master function. It initiates a Read transaction with an addressed slave. Reads one or more bytes (bCnt) from the slave I^2C device and writes data to the array pointed to by pbXferData. Once this routine is called, the included ISR handles further data.

C Prototype:

```
void I2CHW_fReadBytes(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE bMode);
```

Assembler:

```
mov   A,I2CHW_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   A,0x68                 ; Pass slave address 0x68
push  A
lcall  _I2CHW_fReadBytes      ; lcall function to read data from slave
                             ; Reg A contains return value.
add   sp,-5 ; 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 I2CHW_CompleteXfer, a complete transfer is performed. If mode is set to I2CHW_RepStart, or if the mode is set to I2CHW_NoStop, a stop is not generated. This allows an I^2C bus combined transfer to be sent to the slave. Subsequent transfers may then be initiated with a repeat start. See the table at the end of this section.

Return Value:

None.

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.

I2CHW_bWriteBytes

Description:

This is a Master function. It initiates a Write transaction with an addressed slave. Writes one or more bytes (bCnt) to the slave addressed by (bSlaveAddr) from the RAM array pointed to by pbXferData. Once this routine is called, the included ISR handles further data.

C Prototype:

```
void I2CHW_bWriteBytes(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE bMode);
```

Assembler:

```
mov   A,I2CHW_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   A,0x68                 ; Pass slave address 0x68
push  A
lcall  _I2CHW_bWriteBytes     ; lcall function to write data to slave
                             ; Reg A contains return value.
add   sp,-5 ; 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 I2CHW_CompleteXfer, a complete transfer is performed. If mode is set to I2CHW_RepStart, or if the mode is set to I2CHW_NoStop, a stop is not generated. This allows an I^2C bus combined transfer to be sent to the slave. Subsequent turnovers may then be initiated with a repeat start. See the table at the end of this section.

Return Value:

None

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.

I2CHW_bWriteCBytes

Description:

This is a Master function. It initializes a Write transaction with an addressed slave. Writes one or more bytes (bCnt) from a constant flash array (pbXferData) to the slave addressed by bSlaveAddr. Once the data transfer is initiated by this routine, further data transfer is handled by the included ISR.

C Prototype:

```
void I2CHW_bWriteCBytes(BYTE bSlaveAddr, const BYTE * pbXferData, BYTE bCnt, BYTE
bMode);
```

Assembler:

```
mov   A,I2CHW_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   A,0x68                 ; Pass slave address 0x68
push  A
lcall  _I2CHW_bWriteCBytes    ; lcall function to write data to slave
                             ; A contains return value.
add   sp,-5 ; 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 I2CHW_CompleteXfer, a complete transfer is performed. If mode is set to I2CHW_RepStart, or if the mode is set to I2CHW_NoStop, a stop is not generated. This allows an I^2C bus combined transfer to be sent to the slave. Subsequent turnovers may then be initiated with a repeat start. See the table at the end of this section.

Return Value:

None.

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.

Note The bMode parameter may be used to perform an I^2C bus combined format transfer. To execute a combined transfer, first execute an I2CHW_bWriteBytes or I2CHW_bWriteCBytes command with the bMode parameter set to I2CHW_NoStop (0x02). This performs a write without a stop. Next, execute an I2CHW_fReadBytes command with the bMode parameter set to I2CHW_RepStart (0x01).

Constant	Value	Description
I2CHW_CompleteXfer	0x00	Perform complete transfer from Start to Stop
I2CHW_RepStart	0x01	Send Repeat Start instead of Start
I2CHW_NoStop	0x02	Execute transfer without a Stop

I2CHW_bReadBusStatus

Description:

Read the current value of the I2CHW_bStatus byte. This is an internal status byte to the API functions and ISR used for the I2CHW User Module. No facility is given to change the data in theI2CHW_bStatus byte.

C Prototype:

```
BYTE I2CHW_bReadBusStatus (void);
```

Assembler:

```
lcall  I2CHW_bReadBusStatus
```

Parameters:

None

Return Value:

```
BYTE I2CHW_bStatus
```

For detailed description see the following table. These definitions are reporting status variables, not status/control bits.

Constant	Value	Description
RepStart, NoStop	01h, 02h	Reserved for transfer options CompleteXfer/RepStart/NoStop
NAKnextWr	04h	Flag to tell slave to NAK next byte from master

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.

Multimaster Slave Functions

The following functions are specific to the MultiMasterSlave version of the I2CHW User Module.

I2CHW_EnableSlave

Description:

Enable the I^2C Slave function for the I^2C HW block by setting the Enable Slave bit in the I2C_CFG register.

C Prototype:

```
void   I2CHW_EnableSlave(void);
```

Assembler:

```
lcall  I2CHW_EnableSlave
```

Parameters:

None

Return Value:

None

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. As noted in the CY8C27xxxA datasheet, the I^2C configuration registers should be written at 6 MHz CPU clock speed. The routine implemented here does that. If writing directly to the configuration registers, care should be taken to correctly perform the operation or I^2C communication failures may occur.

I2CHW_DisableSlave

Description:

Disables the I^2C Slave function by clearing the Enable Slave bit in the I2C_CFG register.

C Prototype:

```
void   I2CHW_DisableSlave(void);
```

Assembler:

```
lcall  I2CHW_DisableSlave
```

Parameters:

None

Return Value:

None

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. As noted in the CY8C27xxxA datasheet, the I^2C configuration registers should be written at 6 MHz CPU clock speed. The routine implemented here does that. If writing directly to the configuration registers, care should be taken to correctly perform the operation or I^2C communication failures may occur.

I2CHW_EnableMstr

Description:

Enables the I^2C HW block as a Master by setting the Enable Master bit in the I2C_CFG register.

C Prototype:

```
void   I2CHW_EnableMstr(void);
```

Assembler:

```
lcall I2CHW_EnableMstr
```

Parameters:

None

Return Value:

None

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. As noted in the CY8C27xxxA datasheet, the I^2C configuration registers should be written at 6 MHz CPU clock speed. The routine implemented here does that. If writing directly to the configuration registers, care should be taken to correctly perform the operation or I^2C communication failures may occur.

I2CHW_DisableMstr

Description:

Disables the I^2C Master function by clearing the Enable Master bit in the I2C_CFG register.

C Prototype:

```
void   I2CHW_DisableMstr(void);
```

Assembler:

```
lcall  I2CHW_DisableMstr
```

Parameters:

None

Return Value:

None

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. As noted in the CY8C27xxxA datasheet, the I^2C configuration registers should be written at 6 MHz CPU clock speed. The routine implemented here does that. If writing directly to the configuration registers, care should be taken to correctly perform the operation or I^2C communication failures may occur.

I2CHW_InitSlaveWrite

Description:

Initializes a data buffer pointer for the MultiMasterSlave to use in slave mode to deposit data, and initializes the value of a count byte for the same buffer. Count is initialized to the maximum supplied buffer length. On the next instance of a master write, data is placed at the address defined by this function.

C Prototype:

```
void   I2CHW_InitSlaveWrite(BYTE * pWriteBuf, BYTE bBufLen);
```

Assembler:

```
AREA    bss (RAM, REL)
abWriteBuf    blk 10h

AREA    text (ROM,REL)
    push X                      ; save registers
    push A
    add  SP, 3
```

```
    mov  X, SP
    dec  X                          ; X points at data SP points at next
                    ; empty stack location
    mov  [X], <abWriteBuf           ; place the buffer address
    mov  [X-1], >abWriteBuf         ; (page 0)on the stack at [X]
    mov  [X-2], 10                  ; place the count at [x-2]
                                    ; don't care what [X-1] is
                                    ; the compiler would assign 0 as the
                                ; MSB of the Ramtbl addr
    lcall  I2CHW_InitSlaveWrite
    add   SP, -3                    ; restore the stack
    pop   A                         ; restore registers
    pop   X
```

Parameters:

pWriteBuf: Pointer to a RAM buffer location. buf_len: Length of write buffer.

Return Value:

None

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.

I2CHW_InitSlaveRamRead

Description:

Initializes a ram data buffer pointer from which the MultiMasterSlave in slave mode retrieves data, and initializes the value of a count byte for the same buffer. Clears the I2CHW_SlaveStatus flag I2CHW_READFLASH to 0 (Read status bits only), causing the next read to be attempted from a *previously* set buffer location in RAM.

C Prototype:

```
void   I2CHW_InitSlaveRamRead(BYTE * pReadBuf, BYTE bBufLen);
```

Assembler:

```
AREA bss (RAM,REL)
abReadBuf:    blk 10h

AREA text (ROM,REL)
    push  X                         ; save registers
    push  A
add   SP, 3
    mov   X, SP
    dec   X                         ; X points at data SP points at next
                                    ; empty stack location
    mov   [X], <abReadBuf           ; place the read buffer address
    mov   [X-1], >abReadBuf         ; (page0)on the stack at [X]
    mov   [X-2], 10                 ; place the count at [x-2]
                                    ; don't care what [X-1] is
                                    ; the compiler would assign 0 as
                                    ; the MSB of the Ramtbl addr
```

```
    lcall  I2CHW_InitRamRead
    add   SP, -3                    ; back up the stack (subtract 3)
    pop   A                         ; restore registers
    pop   X
```

Parameters:

_ReadBuf: Pointer to a RAM buffer location. bBufLen: Length of read buffer.

Return Value:

None

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.

I2CHW_InitSlaveFlashRead

Description:

Initializes a flash data buffer pointer for retrieval of data. Sets the I2CHW_SlaveStatus flag I2CHW_READFLASH to 1, causing the next read to be attempted from a *previously* set buffer location in flash.

C Prototype:

```
void   I2Cs_InitSlaveFlashRead(const BYTE * pFlashBuf, WORD wBufLen);
```

Assembler:

```
area table(ROM,ABS)
org 0x1015

abFlashBuf:

      db 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88
      db 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff


area text(ROM,REL)


    push  X                         ; save registers
    push  A
    add   SP, 4
    mov   X, SP
    dec   X                         ; X points at data SP points at next
                        ; empty stack location
    mov   [X], <abFlashBuf          ; place the LSB of rom
                        ; address on the stack at [X]
    mov   [X-1], >abFlashBuf        ; place the MSB of the rom address
                        ; at [x-1] variable
    mov   [X-2], 0x0F               ; place the LSB of length
                                    ; at [x-2]
    mov   [X-3], 0x00               ; place the MSB of length
                        ; at [x-3]
```

```
    lcall  I2CHW_InitSlaveFlashRead
    add   SP, -4                    ; adjust the stack (subtract 4)
    pop   A                         ; restore registers
    pop   X
```

Parameters:

pFlashBuf: Pointer to a flash/ROM buffer location. wBufLen: Length of buffer.

Return Value:

None

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.

I2CHW_fReadBytes

Description:

This function initializes a Read transaction with an addressed slave. Reads one or more bytes (bCnt) from the slave I^2C device and writes data to the array pointed to by pbXferData. Once this routine is called, the included ISR handles further data.

C Prototype:

```
void I2CHW_fReadBytes(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE bMode);
```

Assembler:

```
mov   A,I2CHW_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   A,0x68                 ; Pass slave address 0x68
push  A
lcall  I2CHW_fReadBytes       ; lcall function to read data from slave
                             ; Reg A contains return value.
add   sp,-5 ; 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 I2CHW_CompleteXfer, a complete transfer is performed. If mode is set to I2CHW_RepStart, or if the mode is set to I2CHW_NoStop, a stop is not generated. This allows an I^2C bus combined transfer to be sent to the slave. Subsequent transfers may then be initiated with a repeat start. See the table at the end of this section.

Return Value:

None.

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.

I2CHW_bWriteBytes

Description:

This function initializes a Write transaction with an addressed slave. Writes one or more bytes (bCnt) to the slave addressed by (bSlaveAddr) from the RAM array pointed to by pbXferData. Once this routine is called, the included ISR handles further data.

C Prototype:

```
void I2CHW_bWriteBytes(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE bMode);
```

Assembler:

```
mov   A,I2CHW_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   A,0x68                 ; Pass slave address 0x68
push  A
lcall  I2CHW_bWriteBytes      ; lcall function to write data to slave
                             ; Reg A contains return value.
add   sp,-5 ; 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 I2CHW_CompleteXfer, a complete transfer is performed. If mode is set to I2CHW_RepStart, or if the mode is set to I2CHW_NoStop, a stop is not generated. This allows an I^2C bus combined transfer to be sent to the slave. Subsequent transfers may then be initiated with a repeat start. See the table at the end of this section.

Return Value:

None

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.

I2CHW_bWriteCBytes

Description:

This function initializes a Write transaction with an addressed slave. Writes one or more bytes (bCnt) from a constant flash array (pbXferData) to the slave addressed by bSlaveAddr. Once the data transfer is initiated by this routine, further data transfer is handled by the included ISR.

C Prototype:

```
void I2CHW_bWriteCBytes(BYTE bSlaveAddr, const BYTE * pbXferData, BYTE bCnt, BYTE
bMode);
```

Assembler:

```
mov   A,I2CHW_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   A,0x68                 ; Pass slave address 0x68
push  A
lcall  I2CHW_bWriteCBytes     ; lcall function to write data to slave
                             ; A contains return value.
add   sp,-5 ; 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 I2CHW_CompleteXfer, a complete transfer is performed. If mode is set to I2CHW_RepStart, or if the mode is set to I2CHW_NoStop, a stop is not generated. This allows an I^2C bus combined transfer to be sent to the slave. Subsequent transfers may then be initiated with a repeat start. See the table at the end of this section.

Return Value:

None.

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.

Note The bMode parameter may be used to perform an I^2C bus combined format transfer. To execute a combined transfer, first execute an I2CHW_bWriteBytes or I2CHW_bWriteCBytes command with the bMode parameter set to I2CHW_NoStop (0x02). This performs a write without a stop. Next, execute an I2CHW_fReadBytes command with the bMode parameter set to I2CHW_RepStart (0x01).

Constant	Value	Description
I2CHW_CompleteXfer	0x00	Perform complete transfer from Start to Stop
I2CHW_RepStart	0x01	Send Repeat Start instead of Start
I2CHW_NoStop	0x02	Execute transfer without a Stop

I2CHW_bReadSlaveStatus

Description:

Returns the status bits in the Control/Status register.

C Prototype:

```
BYTE  I2CHW_bReadSlaveStatus(void);
```

Assembler:

```
lcall  I2CHW_bReadSlaveStatus ; Accumulator contains status
```

Parameters:

None

Return Value:

```
BYTE  I2CHW_SlaveStatus
```

See Table.

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.

Constant	Value	Description
I2CHW_RD_NOERR	01h	Data read by the master, normal ISR exit
I2CHW_RD_ OVERFLOW	02h	More data bytes were read by the master than were available
I2CHW_RD_ COMPLETE	04h	A read was initiated and has been completed
I2CHW_READFLASH	08h	The next read is from a flash location
I2CHW_WR_NOERR	10h	Data was written successfully by the master
I2CHW_WR_OVERFLOW	20h	The master wrote too many bytes for the write buffer
I2CHW_WR_COMPLETE	40h	A master write was completed by a new address or stop
I2CHW_ISR_ACTIVE	80h	The I^2C ISR has not yet exited and is active

I2CHW_ ClrSlaveRdStatus

Description:

Clears the read status bits in the I2CHW_SlaveStatus register. No other bits are affected.

C Prototype:

```
void  I2CHW_ClrSlaveRdStatus (void);
```

Assembler:

```
lcall  I2CHW_ClrSlaveRdStatus
```

Parameters:

None

Return Value:

None

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.

I2CHW_ ClrSlaveWrStatus

Description:

Clears the write status bits in the I2CHW_SlaveStatus register. No other bits are affected.

C Prototype:

```
void  I2CHW_ClrSlaveWrStatus (void);
```

Assembler:

```
lcall  I2CHW_ClrSlaveWStatus
```

Parameters:

None

Return Value:

None

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.

I2CHW_bReadMasterStatus

Description:

Returns the status bits in the Control/Status register.

C Prototype:

```
BYTE  I2CHW_bReadMasterStatus(void);
```

Assembler:

```
lcall  I2CHW_bReadMasterStatus ; Accumulator contains status
```

Parameters:

None

Return Value:

```
BYTE  I2CHW_MasterStatus
```

See Table.

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.

Constant	Value	Description
I2CHW_RD_NOERR	01h	Data read by the master, normal ISR exit
I2CHW_RD_ OVERFLOW	02h	More data bytes were read by the master than were available
I2CHW_RD_ COMPLETE	04h	A read was initiated and has been completed
I2CHW_READFLASH	08h	The next read is from a flash location
I2CHW_WR_NOERR	10h	Data was written successfully by the master
I2CHW_WR_OVERFLOW	20h	The master wrote too many bytes for the write buffer
I2CHW_WR_COMPLETE	40h	A master write was completed by a new address or stop
I2CHW_ISR_ACTIVE	80h	The I^2C ISR has not yet exited and is active

I2CHW_ ClrMasterRdStatus

Description:

Clears the read status bits in the I2CHW_MasterStatus register. No other bits are affected.

C Prototype:

```
void  I2CHW_ClrMasterRdStatus (void);
```

Assembler:

```
lcall  I2CHW_ClrMasterRdStatus
```

Parameters:

None

Return Value:

None

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.

I2CHW_ ClrMasterWrStatus

Description:

Clears the write status bits in the I2CHW_MasterStatus register. No other bits are affected.

C Prototype:

```
void  I2CHW_ClrMasterWrStatus (void);
```

Assembler:

```
lcall  I2CHW_ClrMasterWStatus
```

Parameters:

None

Return Value:

None

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.

I2CHW_bReadBusStatus

Description:

Read the current value of the I2CHW_bStatus byte. This is an internal status byte to the API functions and ISR used for the I2CHW User Module. No facility is given to change the data in theI2CHW_bStatus byte. This byte is used by all 3 versions of the I2CHW User Module but only to given information to the MultiMasterSlave version because of the additional status information given as a consequence of operation in a MultiMasterSlave environment.

C Prototype:

```
BYTE I2CHW_bReadBusStatus (void);
```

Assembler:

```
lcall  I2CHW_bReadBusStatus
```

Parameters:

None

Return Value:

```
BYTE I2CHW_bStatus
```

For detailed description see table. Those definitions are reporting status variables, not status/control bits.

Constant	Value	Description
RepStart, NoStop	01h, 02h	Reserved for transfer options CompleteXfer/RepStart/NoStop
BUS_BUSY	04h	The bus is busy
LOST_ARB	08h	The master has lost arbitration
BUS_ERROR	10h	A bus error has occurred
SLAVE_NAK	20h	A slave has failed to respond
ERROR	40h	A requested operation failed
ISR_ACTIVE	80h	ISR is active and I2C operation

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.

I2CHW_fReadBytesNoStall

Description:

This function initializes a Read transaction with an addressed slave. Reads one or more bytes (bCnt) from the slave I2C device and writes data to the array pointed to by pbXferData. Once this routine is called, the included ISR handles further data as long as the bus is not busy servicing another master.

C Prototype:

```
BYTE I2CHW_fReadBytesNoStall(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE
bMode);
```

Assembler:

```
mov A,I2CHW_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 A,0x68 ; Pass slave address 0x68
push A
lcall I2CHW_fReadBytesNoStall ; lcall function to read data from slave
; Reg A contains return value.
add sp,-5 ; 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 I2CHW_CompleteXfer, a complete transfer is performed. If mode is set to I2CHW_RepStart, or if the mode is set to I2CHW_NoStop, a stop is not generated. This allows an I2C bus combined transfer to be sent to the slave. Subsequent transfers may then be initiated with a repeat start. See the table at the end of this section.

Return Value:

BYTE I2CHW_MasterStatus if bus was not busy or 0xFF if the bus was busy.

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.

I2CHW_bWriteBytesNoStall

Description:

This function initializes a Write transaction with an addressed slave. Writes one or more bytes (bCnt) to the slave addressed by (bSlaveAddr) from the RAM array pointed to by pbXferData. Once this routine is called, the included ISR handles further data as long as the bus is not busy servicing another master.

C Prototype:

```
BYTE I2CHW_bWriteBytesNoStall(BYTE bSlaveAddr, BYTE * pbXferData, BYTE bCnt, BYTE
bMode);
```

Assembler:

```
mov A,I2CHW_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 A,0x68 ; Pass slave address 0x68
push A
lcall I2CHW_bWriteBytesNoStall ; lcall function to write data to slave
; Reg A contains return value.
add sp,-5 ; 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 I2CHW_CompleteXfer, a complete transfer is performed. If mode is set to I2CHW_RepStart, or if the mode is set to I2CHW_NoStop, a stop is not generated. This allows an I2C bus combined transfer to be sent to the slave. Subsequent transfers may then be initiated with a repeat start. See the table at the end of this section.

Return Value:

BYTE I2CHW_MasterStatus if bus was not busy or 0xFF if the bus was busy.

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.

I2CHW_bWriteCBytesNoStall

Description:

This function initializes a Write transaction with an addressed slave. Writes one or more bytes (bCnt) from a constant flash array (pbXferData) to the slave addressed by bSlaveAddr. Once the data transfer is initiated by this routine, further data transfer is handled by the included ISR as long as the bus is not busy servicing another master.

C Prototype:

```
BYTE I2CHW_bWriteCBytesNoStall(BYTE bSlaveAddr, const BYTE * pbXferData, BYTE bCnt,
BYTE bMode);
```

Assembler:

```
mov A,I2CHW_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
mov A,0x68 ; Pass slave address 0x68
push A
lcall I2CHW_ bWriteCBytesNoStall ; lcall function to write data to slave
; A contains return value.
add sp,-5 ; 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 I2CHW_CompleteXfer, a complete transfer is performed. If mode is set to I2CHW_RepStart, or if the mode is set to I2CHW_NoStop, a stop is not generated. This allows an I2C bus combined transfer to be sent to the slave. Subsequent transfers may then be initiated with a repeat start. See the table at the end of this section.

Return Value:

BYTE I2CHW_MasterStatus if bus was not busy or 0xFF if the bus was busy.

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.

Note The bMode parameter may be used to perform an I2C bus combined format transfer. To execute a combined transfer, first execute an I2CHW_bWriteBytes or I2CHW_bWriteCBytes command with

the bMode parameter set to I2CHW_NoStop (0x02). This performs a write without a stop. Next, execute an I2CHW_fReadBytes, or I2CHW_fReadBytesNoStall command if there are other masters on the bus, with the bMode parameter set to I2CHW_RepStart (0x01).

Constant	Value	Description
I2CHW_RepStart	01h	Internal status bit used to control stop/repeat start of Master
I2CHW_NoStop	02h	Internal status used to control stop/repeat start of Master
I2CHW_BUS_BUSY	04h	A flag indicating that the I2C bus is in use by this device
I2CHW_LOST_ARB	08h	Status flag to indicate the master lost arbitration to another master and did NOT acquire control of the I2C bus
I2CHW_BUS_ERROR	10h	An illegal condition was detected on the I2C bus
I2CHW_SLAVE_NAK	20h	An external slave addressed by the this Master has NAK'ed it's address
I2CHW_ERROR	40h	This Master attempted an operation that was unsuccessful
I2CHW_ISR_ACTIVE	80h	Some sort of Master or Slave activity is currently in process for this device.

I2CHW_ EnableHWAddrCheck

Description:

This function applies to only to the CY8C21x45 and CY8C22x45 device families. It enables the hardware address comparison feature.

C Prototype:

```
void  I2CHW_EnableHWAddrCheck (void);
```

Assembler:

```
lcall  I2CHW_EnableHWAddrCheck
```

Parameters:

None

Return Value:

None

Side Effects:

Make sure that if the alternative slave address (flash registers) is enabled, the device does not respond to it while the hardware address comparison feature is active. This is because the I2C slave responds only to its primary address.

I2CHW_ DisableHWAddrCheck

Description:

This function applies to only to the CY8C21x45 and CY8C22x45 device families. It disables the hardware address comparison feature.

C Prototype:

```
void  I2CHW_DisableHWAddrCheck (void);
```

Assembler:

```
lcall  I2CHW_DisableHWAddrCheck
```

Parameters:

None

Return Value:

None

Side Effects:

Make sure that if the alternative slave address (flash registers) is enabled, the device does not respond to it while the hardware address comparison feature is active. This is because the I2C slave responds only to its primary address.

Sample Firmware Source Code

Here is an implementation of the I2CHW User Module configured as a slave:

```
/*****************************************************************************/
/* Sample assembly code to commuinication with:                              */
/*  C Master sample code.                                                    */
/*  ASM Master sample code.                                                  */
/*                                                                           */
/* This code writes and reads back echoed data from the slave.               */
/*                                                                           */
/* NOTE: 1. I2CHW does not depend on the CPU clock.                          */
/*       2. The instance name of the I2CHWs User Module is assumed to        */
/*          be I2CHW.                                                        */
/*       3. Device is assumed to be Large Memory Model.                      */
/*****************************************************************************/
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

/* Define buffer size */
#define BUFFERSIZE 8
/* Setup a 8 byte buffer */
BYTE txRxBuffer[BUFFERSIZE];
BYTE status;

void I2C_poll(void)
{
    status = I2CHW_bReadI2CStatus();
        /* Wait to read data from the master */
        if( status & I2CHW_WR_COMPLETE )
        {
            /* Data received - clear the Write status */
            I2CHW_ClrWrStatus();
            /* Reset the pointer for the next read data */
            I2CHW_InitWrite(txRxBuffer,BUFFERSIZE);
        }
        /* wait until data is echoed */
        if( status & I2CHW_RD_COMPLETE )
        {
            /* Data echoed - clear the read status */
```

```
            I2CHW_ClrRdStatus();
            /* Reset the pointer for the next data to echo */
            I2CHW_InitRamRead(txRxBuffer,BUFFERSIZE);
        }
}
void main(void)
{
    /* Start the slave and wait for the master */
    I2CHW_Start();
    I2CHW_EnableSlave();
    /* Enable the global and local interrupts */
    M8C_EnableGInt;
    I2CHW_EnableInt();
    /* Setup the Read and Write Buffer - set to the same buffer */
    I2CHW_InitRamRead(txRxBuffer,BUFFERSIZE);
    I2CHW_InitWrite(txRxBuffer,BUFFERSIZE);
    /* Echo forever */
    while( 1 )
    {
        I2C_poll();

        // Place user code here to update and read structure data.
        // Please note that the I2C_poll() should be called often enough to properly
        //serve I2C transactions.
        // Thus if user code is large call the I2C_poll() multiple times during main
        //loop execution.
    }
} //end of main
```

Implementation of the I2CHW User Module configured as a master:

```
/*****************************************************************************/
/* Sample assembly code to commuinication with:                              */
/*  C Slave sample code.                                                     */
/*  ASM Slave sample code.                                                   */
/*                                                                           */
/* This sample code will transmit data to a slave and then will read         */
/* back from the slave.                                                      */
/*                                                                           */
/* NOTE: 1. I2CHW does not depend on the CPU clock.                          */
/*       2. The instance name of the I2CHWm User Module is assumed to        */
/*          be I2CHW.                                                        */
/*       3. Device is assumed to be Large Memory Model.                      */
/*****************************************************************************/
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

/* Define slave address */
#define SLAVE_ADDRESS 0x55
/* Define buffer size */
#define BUFFER_SIZE 0x08

/* Setup buffers */
BYTE txBuffer[BUFFER_SIZE];
BYTE rxBuffer[BUFFER_SIZE];
```

```
BYTE status;

void main(void)
{
    /* Start the master */
    I2CHW_Start();
    I2CHW_EnableMstr();
    /* Enable the global and local interrupts */
    M8C_EnableGInt;
    I2CHW_EnableInt();

/* Send and Receive forever*/
while( 1 )
    {
        /* Send the contents of the data in txBuffer */;
        I2CHW_bWriteBytes(SLAVE_ADDRESS, txBuffer, BUFFER_SIZE, I2CHW_CompleteXfer);
        /* Wait until the data is transferred */
        while(!(I2CHW_bReadI2CStatus() & I2CHW_WR_COMPLETE));
        /* Clear Write Complete Status bit */
        I2CHW_ClrWrStatus();

/* Read from the slave and place in rxBuffer */;
        I2CHW_fReadBytes(SLAVE_ADDRESS, rxBuffer, BUFFER_SIZE, I2CHW_CompleteXfer);
        /* Wait until the data is read */
        while(!(I2CHW_bReadI2CStatus() & I2CHW_RD_COMPLETE));
        /* Clear Read Complete Status bit */
        I2CHW_ClrRdStatus();

/* Increment 1st byte in txBuffer so the data changes each loop */
        txBuffer[0]++;
    }
} //end of main
```

Implementation of the I2CHW User Module configured as a MultiMasterSlave:

```
/*****************************************************************************/
/* This sample code will receive data from Muster and                         */
/* transmit data to a Slave.                                                 */
/*                                                                           */
/* The instance name of the I2CHWs User Module is assumed                    */
/* to be I2CHW.                                                              */
/*                                                                           */
/* NOTE: 1. I2CHW does not depend on the CPU clock.                          */
/*       2. The instance name of the I2CHWm User Module is assumed to        */
/*          be I2CHW.                                                       */
/*       3. Device is assumed to be Large Memory Model.                      */
/*****************************************************************************/
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

/* Define Slave address */
#define SLAVEADDRESS 0x7

/* Define buffer size */
#define BUFFERSIZE 0x10
```

```
void main(void)
{
    /* Setup buffer */
    BYTE TxRxBuffer[BUFFERSIZE];

/* Start the slave and master */
    I2CHW_Start();
    I2CHW_EnableSlave();
    I2CHW_EnableMstr();

/* Setup the read and write buffers */
    I2CHW_InitSlaveRamRead(TxRxBuffer,BUFFERSIZE);
    I2CHW_InitSlaveWrite(TxRxBuffer,BUFFERSIZE);

/* Enable the local and global interrupts */
    I2CHW_EnableInt();
    M8C_EnableGInt;

/* Loop forever */
    while(1)
    {
        /* Look for data write from master */
        if (I2CHW_bReadSlaveStatus() & I2CHW_WR_COMPLETE)
        {
            /* Data received - clear the Write status */
            I2CHW_ClrSlaveWrStatus();

            /* Reset the pointer for the next write data */
            I2CHW_InitSlaveWrite(TxRxBuffer,BUFFERSIZE);

            /* Master sends the content of the data in Buffer to the Slave */
          I2CHW_bWriteBytes(SLAVEADDRESS, TxRxBuffer, BUFFERSIZE, I2CHW_CompleteXfer);

            /* Wait until the data is transferred */
            while(!(I2CHW_bReadMasterStatus() & I2CHW_WR_COMPLETE));

            /* Clear Write Complete Status bit */
            I2CHW_ClrMasterWrStatus();
        }

        /* If any Master has read data, it is necessary to reset the read pointer
        and clear the Read Complete Status bit for the correct next read of data */
        if (I2CHW_bReadSlaveStatus() & I2CHW_RD_COMPLETE)
        {
            /* Clear Read Complete Status bit */
            I2CHW_ClrSlaveRdStatus();

/* Reset the Read pointer */
            I2CHW_InitSlaveRamRead(TxRxBuffer,BUFFERSIZE);
        }

/* uncomment this code for Errors handling */
        // if (I2CHW_SlaveStatus & (I2CHW_WR_OVERFLOW | I2CHW_RD_OVERFLOW))
        //{
```

```
        //    I2CHW_ClrSlaveWrStatus();
         //   I2CHW_ClrSlaveRdStatus();
        //    I2CHW_InitSlaveRamRead(buf_wr,2);
        //    I2CHW_InitSlaveWrite(buf_wr,2);
        //}
    }
}
```

Implementation of the I2CHW User Module configured as a slave written in assembly code:

```
;*****************************************************************************
;* Sample assembly code to commuinication with:                              *
;*  C Master sample code.                                                    *
;*  ASM Master sample code.                                                  *
;*                                                                           *
;* This code writes and reads back echoed data from the slave.               *
;*                                                                           *
;* NOTE: 1. I2CHW does not depend on the CPU clock.                          *
;*       2. The instance name of the I2CHWm User Module is assumed to        *
;*          be I2CHW.                                                        *
;*       3. Device is assumed to be Large Memory Model.                      *
;*****************************************************************************
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

BUFFERSIZE: equ 8

export txRxBuffer

area bss(RAM)

txRxBuffer: blk BUFFERSIZE

area text(ROM, REL)

export _main

_main:
    ; Initialize I2CHW
    lcall I2CHW_Start
    lcall I2CHW_EnableSlave
    ; Enable the global and local interrupts
    M8C_EnableGInt
    ; Enable the I2CHW as an ISR based process
    lcall I2CHW_EnableInt

; Set the Read Buffer
    mov A, BUFFERSIZE ; Pass the byte count
   push A
    mov A,>txRxBuffer ; Load the MSB of the rxBuffer pointer
    push A
    mov A,<txRxBuffer ; Load the LSB of the rxBuffer pointer
    push A
    lcall I2CHW_InitRamRead
```

```
    add SP, -3 ; Restore the stack

; Set the Write Buffer
    mov A, BUFFERSIZE ; Pass the byte count
    push A
    mov A,>txRxBuffer ; Load the MSB of the rxBuffer pointer
    push A
    mov A,<txRxBuffer ; Load the LSB of the rxBuffer pointer
    push A
    lcall I2CHW_InitWrite
    add SP,-3 ; Restore the stack

    ; Echo forever
    ; Wait until some data is transferred
    CheckI2CStatus:
        lcall I2CHW_bReadI2CStatus ; Accumulator contains status
        push A ; Preserve a copy of A for RD comparison

; Look for data read from master
        and A,I2CHW_RD_COMPLETE
        jnz ReadHappened
        ; Look for data write from master
pop A ; Retrieve preserved copy of A
        and A,I2CHW_WR_COMPLETE
        jnz WriteHappened

    jmp CheckI2CStatus

    ReadHappened:
        ; Clears the read status bits in the I2CHW_RsrcStatus register
        lcall I2CHW_ClrRdStatus
        ;Reset the pointer for the next data to echo
        mov A,BUFFERSIZE ; Pass the byte count
        push A
        mov A,>txRxBuffer ; Load the MSB of the rxBuffer pointer
        push A
        mov A,<txRxBuffer ; Load the LSB of the rxBuffer pointer
        push A
        lcall I2CHW_InitRamRead
        add SP,-4 ; Restore the stack
jmp CheckI2CStatus

    WriteHappened:
        ; Clears the write status bits in the I2CHW_RsrcStatus register
        lcall I2CHW_ClrWrStatus
        ;Reset the pointer for the next data write
        mov A,BUFFERSIZE ; Pass the byte count
        push A
        mov A,>txRxBuffer ; Load the MSB of the rxBuffer pointer
        push A
        mov A,<txRxBuffer ; Load the LSB of the rxBuffer pointer
        push A
        lcall I2CHW_InitWrite
        add SP,-3 ; Restore the stack
        jmp CheckI2CStatus
```

```
;end _main
```

Implementation of the I2CHW User Module configured as a master written in assembly code:

```
;*****************************************************************************
;* Sample assembly code to commuinication with:                              *
;*  C Slave sample code.                                                     *
;*  ASM Slave sample code.                                                   *
;*                                                                           *
;* This sample code will transmit data to a slave and then will read         *
;* back from the slave.                                                      *
;*                                                                           *
;* NOTE: 1. I2CHW does not depend on the CPU clock.                          *
;*       2. The instance name of the I2CHWm User Module is assumed to        *
;*          be I2CHW.                                                        *
;*       3. Device is assumed to be Large Memory Model.                      *
;*****************************************************************************
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

BUFFERSIZE: equ 0x08 ; Define buffer size
SLAVEADDRESS: equ 0x55 ; Define slave address

export txBuffer
export rxBuffer

area bss(RAM)

txBuffer: blk BUFFERSIZE
rxBuffer: blk BUFFERSIZE

area text(ROM, REL)

export _main

_main:
    ; Initialize I2CHW
    lcall I2CHW_Start
    lcall I2CHW_EnableMstr
    ; Enable the global and local interrupts
    M8C_EnableGInt
   ; Enable the I2CHW as an ISR based process
    lcall I2CHW_EnableInt

    ; Send the contents of the data in txBuffer
    WriteTXBuffer:
        mov A,I2CHW_CompleteXfer ; Pass complete transfer flag
        push A
        mov A,BUFFERSIZE ; Pass the byte count
        push A
        mov A,>txBuffer ; Load the MSB of the txBuffer pointer
        push A
        mov A,<txBuffer ; Load the LSB of the txBuffer pointer
```

```
        push A
        mov A,SLAVEADDRESS ; Pass slave address SLAVEADDRESS
        push A
        ; Call function to write data to slave
        lcall _I2CHW_bWriteBytes
        ; Reg A contains return value.
        add SP,-5 ; Restore the stack
        jmp CheckI2CStatus

    ; Read from the slave and place in rxBuffer
    ReadRXBuffer:
        mov A,I2CHW_CompleteXfer ; Pass complete transfer flag
        push A
        mov A,BUFFERSIZE ; Pass the byte count
        push A
        mov A,>rxBuffer ; Load the MSB of the rxBuffer pointer
        push A
        mov A,<rxBuffer ; Load the LSB of the rxBuffer pointer
        push A
        mov A,SLAVEADDRESS ; Pass slave address SLAVEADDRESS
        push A
        ; Call function to read data from slave
        lcall _I2CHW_fReadBytes
        ; Reg A contains return value.
        add SP,-5 ; Restore the stack
        jmp CheckI2CStatus ; Poll I2C status

    ; Wait until the data is transferred
    CheckI2CStatus:
        lcall I2CHW_bReadI2CStatus ; Accumulator contains status
        push A ; Preserve a copy of A for RD comparison
        and A, I2CHW_WR_COMPLETE
       jnz WriteHappened
        pop A ; Retrieve preserved copy of A
        and A, I2CHW_RD_COMPLETE
        jnz ReadHappened
        jmp CheckI2CStatus

    ReadHappened:
        ; Clears the read status bits in the I2CHW_RsrcStatus register
        lcall I2CHW_ClrRdStatus
        ; Do something after read
       ; Increment 1st byte in txBuffer so the data changes each loop
        inc [txBuffer+0]
        jmp WriteTXBuffer

    WriteHappened:
        ; Clears the write status bits in the I2CHW_RsrcStatus register
        lcall I2CHW_ClrWrStatus
        ; Do something with the data
        jmp ReadRXBuffer

;end _main
```

Implementation of the I2CHW User Module configured as a MultiMasterSlave written in assembly code:

```
;*****************************************************************************
;* This sample code will receive data from Muster and                        *
;*  transmit data to a slave.                                                *
;*                                                                           *
;* NOTE: 1. I2CHW does not depend on the CPU clock.                          *
;*       2. The instance name of the I2CHWm User Module is assumed to        *
;*          be I2CHW.                                                        *
;*       3. Device is assumed to be Large Memory Model.                      *
;*****************************************************************************
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

; Define slave address
SLAVEADDRESS:  equ 0x7

; Define buffer size
BUFFERSIZE:    equ 0x10

area bss(RAM)

; Setup buffer
TxRxBuffer:     blk BUFFERSIZE

area text(ROM, REL)

export _main

_main:
    ; Start the slave and master
    lcall I2CHW_Start
    lcall I2CHW_EnableSlave
    lcall I2CHW_EnableMstr

    ; Setup the read and write buffers of the Slave
    ; Set the read buffer
    mov A, BUFFERSIZE ; Pass the byte count
    push A
    mov A, >TxRxBuffer ; Load the MSB of the TxRxBuffer pointer
    push A
    mov A, <TxRxBuffer ; Load the LSB of the TxRxBuffer pointer
    push A
    lcall I2CHW_InitSlaveRamRead
    add SP, -3 ; Restore the stack

    ; Set the write buffer
    mov A, BUFFERSIZE ; Pass the byte count
    push A
    mov A,>TxRxBuffer ; Load the MSB of the TxRxBuffer pointer
    push A
    mov A,<TxRxBuffer ; Load the LSB of the TxRxBuffer pointer
    push A
    lcall I2CHW_InitSlaveWrite
    add SP,-3 ; Restore the stack
```

```
    ; Enable the local and global interrupts
    lcall I2CHW_EnableInt
    M8C_EnableGInt

; loop forever
loop:

    ; Look for data write from master
    lcall I2CHW_bReadSlaveStatus ; Accumulator contains status
    push A ; Preserve a copy of A for RD comparison
    and A,  I2CHW_WR_COMPLETE
    jz ReadHappened

; Data received - clear the write status
lcall I2CHW_ClrSlaveWrStatus

; Reset the write pointer for the next write data
mov A, BUFFERSIZE ; Pass the byte count
    push A
    mov A,>TxRxBuffer ; Load the MSB of the TxRxBuffer pointer
    push A
    mov A, <TxRxBuffer ; Load the LSB of the TxRxBuffer pointer
    push A
    lcall I2CHW_InitSlaveWrite
    add SP,-3 ; Restore the stack

; Master sends the content of the data in Buffer to the Slave
mov A, I2CHW_CompleteXfer ; Pass complete transfer flag
    push A
    mov A,BUFFERSIZE ; Pass the byte count
    push A
    mov A,>TxRxBuffer ; Load the MSB of the TxRxBuffer pointer
    push A
    mov A, <TxRxBuffer ; Load the LSB of the TxRxBuffer pointer
    push A
    mov A,SLAVEADDRESS ; Pass slave address SLAVEADDRESS
    push A
    lcall _I2CHW_bWriteBytes; Reg A contains return value
    add SP,-5 ; Restore the stack

; Wait until the data is transferred
CheckI2CStatus:
    lcall I2CHW_bReadMasterStatus ; Accumulator contains status
    and A, I2CHW_WR_COMPLETE
    jz CheckI2CStatus

    ; Clear Write Complete Status bit
lcall I2CHW_ClrMasterWrStatus

; If any Master has read data, it is necessary to reset the read pointer
; and clear the Read Complete Status bit for the correct next read of data
ReadHappened:
    pop A ; Retrieve preserved copy of A
    and A,I2CHW_RD_COMPLETE
    jz ErrorHappened
```

```
; Clear Read Complete Status bit
lcall I2CHW_ClrSlaveRdStatus

; Reset the Read pointer
    mov A, BUFFERSIZE ; Pass the byte count
    push A
    mov A, >TxRxBuffer ; Load the MSB of the TxRxBuffer pointer
    push A
    mov A, <TxRxBuffer ; Load the LSB of the TxRxBuffer pointer
    push A
    lcall I2CHW_InitSlaveRamRead
    add SP, -3 ; Restore the stack

ErrorHappened:
    ;insert here code for Errors handling

jmp loop
```

Configuration Registers

This section describes the PSoC Resource Registers used or modified by the I^{2}C HW User Module.

Table 1. Resource I2C_CFG: Bank 0 reg[D6] Configuration Register

Bit	7	6	5	4	3	2	1	0
Value	Reserved	PinSelect	Bus Error IE	Stop IE	Clock Rate[1]	Clock Rate[0]	Enable Master	Enable Slave

Pin Select: Selects either SCL and SDA as P1[5]/P1[7] or P1[0]/P1[1].

Bus Error Interrupt Enable: Enable I^{2}C interrupt generation on a Bus Error.

Stop Error Interrupt Enable: Enable an I^{2}C interrupt on an I^{2}C Stop condition.

Clock Rate[1,0]: Select among 3 valid Clock rates 50- 100- 400 kbps.

Enable Master: Enable the I^{2}C HW block as a bus Master.

Enable Slave: Enable the I^{2}C HW block as a bus Slave.

Table 2. Resource I2C_SCR: Bank 0 reg[D7] Status Control Register

Bit	7	6	5	4	3	2	1	0
Value	Bus Error	Lost Arb	Stop Status	ACK out	Address	Transmit	Last Recd Bit (LRB)	Byte Complete

Bus Error: Indicates a Bus Error condition has been detected.

Lost Arbitration: In MultiMaster mode indicates loss of arbitration for this device, (device does not control bus).

Stop Status: An I^{2}C stop condition has been detected.

ACK out: direct the I^2C block to Acknowledge (1) or Not Acknowledge (0) a received byte.

Address: Received or transmitted byte is an address.

Transmit: Transmit bit is set by the firmware to define the direction of the byte transfer. Any Start detect or write to the Start or Restart bit, when operating in Master mode will clear this bit. 0-Receive Mode, 1-Transmit Mode.

Last Received Bit (LRB): Value of last received bit (bit 9) in a transmit sequence, status of Ack/Nak from destination device.

Byte Complete: 8 data bits have been received. For Receive Mode, the bus is stalled waiting for an Ack/Nak. For Transmit Mode Ack Nak has also been received (see LRB) and the bus is stalled for the next action to be taken

Table 3. Resource I2C_DR: Bank 0 reg[D8] Data Register

Bit	7	6	5	4	3	2	1	0
Value	Data							

Received or Transmitted data. To transmit data, this register must be loaded before a write to the I2C_SCR register. Received data is read from this register. It may contain an address or data.

Table 4. Resource I2C_MSCR: Bank 0 reg[D9] Master Status Control Register

Bit	7	6	5	4	3	2	1	0
Value	Reserved	Reserved	Reserved	Reserved	Bus Busy	Master Mode	Restart Gen	Start Gen

Bus Busy: Master Only, set when any bus Start condition is detected, cleared when a Stop condition is detected.

Master Mode: Indicates the device is currently operating as a bus Master.

Restart Gen: Master only, may be set to generate a repeat start for the I^2C bus.

Start Gen: Master Only, When bus becomes idle, generate an I^2C bus start and transmit an I^2C address using data in the data register (I2C_DR).

Version History

Version	Originator	Description
1.6	DHA	Added Version History.
1.7	DHA	The following changes were done to the Start function: 1. Changed Initial Open-Drain Low drive mode of user module pins to HI-Z analog. 2. Enabled the I^2C block. 3. Gave delay 5 nop instructions. 4. Restored the Initial I^2C pin drive mode.
1.80	DHA	Added I2CHW_bReadBusStatus() function for Master configuration. Synchronized API function definitions in header files with datasheet. Reorganized precompiler directives in .inc and .asm files.
1.80.b	DHA	Comment style changed from ':' to '//'.
1.90	DHA	Updated the variable name "@INSTANCE_NAME`_DoBufferRepeatStart" to support two I2C user module instances on the CY8C28X45 device.
1.90.b	DHA	1. Added a note to the I2C Clock parameter description in this user module datasheet about Clock Dependency on SYSCLK. 2. Updated the DC and AC Electrical Characteristics section in this user module datasheet.
1.90.c	DHA	Updated Sample firmware code slave mode in the user module datasheet.
2.00	MYKZ	1. Corrected method of clearing posted interrupts. 2. Added support of hardware address detection on the CY8C2xx45 family.

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.

Copyright © 2003-2015 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.