SMBus Slave Datasheet SMBusSlave V 2.00

Resources	PSoC® Blocks			API Memory (Bytes)		Pins
	Digital	Analog CT	Analog SC	Flash	RAM	
Supported devices: CY8C29x66, CY8C27x43, CY8C28xxx, CY8C24x23, CY8C24x33, CY8C21x23, CY8C21x34, CY8C21x45, CY8C22x45, CY8C24x94						
Configuration 1	0	0	0	934	31	2–5

Features and Overview

- Industry standard SMBus compatible slave interface
- Supports SMBus Host Notify Protocol (HNP) using command and through a dedicated Alert pin
- Optional Packet Error Check (PEC)
- Standard data rate of 50/100 kbps
- High-level Application Program Interface (API) requires minimal user programming

The SMBus Slave User Module provides a SMBus slave interface that is fully compliant with the Physical and Digital Link layers described in the SMBus Specification Version 2.0. This user module can be used in conjunction with other master and slave devices connected to a single SMBus segment. This user module uses an I2C controller and its interrupt for its physical layer.

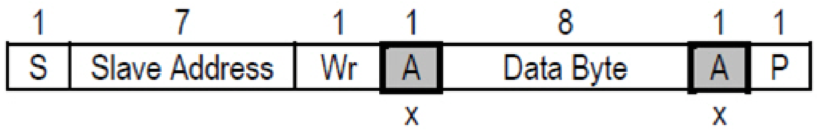
Functional Description

The SMBus Slave User Module communicates with the bus master using the protocol described in the Data Link layer of the SMBus Specification Version 2.0. It communicates over the SMBDAT and SMBCLK lines and has the following features:

- Detects START, repeated START, STOP, and bus idle conditions
- Responds with an ACK when addressed by the bus master
- Achieves clock-stretching and synchronization
- Communicates in accordance with all of the specification's bus protocols
- Responds to data with ACK/NACK
- Implements PEC when specified by the user
- Notifies the host using the SMBus HNP

The SMBus Slave User Module supports all of the bus protocols mentioned in Section 5.5 of the SMBus Specification Version 2.0. For all of these protocols, PEC applies if the Packet Error Check parameter is enabled. The generic protocol diagram is shown in Figure 1.

Figure 1. Generic Protocol Diagram



S	Start Condition
Sr	Repeated Start Condition
Rd	Read (bit value of 1)
Wr	Write (bit value of 0)
x	Shown under a field indicates that that field is required to have the value of 'x'
A	Acknowledge (this bit position may be '0' for an ACK or '1' for a NACK)
P	Stop Condition
PEC	Packet Error Code
☐	Master-to-Slave
▨	Slave-to-Master
...	Continuation of protocol

Note The SMBus Slave User Module does not support Address Resolution Protocol (ARP).

Addressing Mechanism

The slave address has to be configured in the parameters wizard using the Slave Address parameter. It can also be dynamically changed in firmware using SMBusSlave_SetAddr().

You can use two GPIO pins to set the lower two bits of the slave address by enabling the Hardwired Slave Address parameter. The Slave Address Pin A0 and Slave Address Pin A1 parameters select the two GPIO pins. If the Hardwired Slave Address parameter is enabled, these two pins will be monitored during startup, and their logic levels will be used to set the two least significant bits of the slave address (Pin A1 = slave address bit 1, Pin A0 = slave address bit 0). The upper five bits of the slave address will equal bits 6–2 of the Slave Address parameter.

When SMBusSlave_SetAddr() is used to dynamically change the slave address in firmware, Slave Address Pin A0 and Slave Address Pin A1 are ignored even if the Hardwired Slave Address parameter is enabled. The new slave address will be set by the parameters passed to the SMBusSlave_SetAddr() function. Slave Address Pin A0 and Slave Address Pin A1 are only monitored during startup.

RAM Interface for SMBus Commands

The SMBus Slave User Module allows the master to access specific RAM buffers through each SMBus command. The data presented to the master can be a single variable, an array of values, or a structure. A different RAM buffer can be exposed for each SMBus command. The SMBus Slave User Module will read from/write to the appropriate RAM buffer depending on the command it receives from the master.

Before any transfers occur, the user has to assign a command code and initialize the RAM buffer for each SMBus command using SMBusSlave_SetCommand() and SMBusSlave_SetBuffer(). Both of these functions have to be performed during initialization before starting the user module.

Assign a Command Code

SMBusSlave_SetCommand (BYTE Command_Name, BYTE Command_Code)

Command Name: The SMBus command name that the user module uses to map the command name to its command code. Possible values for this parameter are: WRITE_BYTE, WRITE_WORD, READ_BYTE, READ_WORD, PROCESS_CALL, BLOCK_WRITE, BLOCK_READ, and BLOCK_WRITE_BLOCK_READ_PROCESS_CALL.

Command Code: The command code for the given command name. Possible values for this parameter are 0x00-0xFF. The user has to make sure no two commands have the same command code.

Exceptions: The Quick Command, Send Byte, and Receive Byte commands do not need to be assigned a command code.

Initialize the RAM Buffer

SMBusSlave_ SetBuffer (BYTE Command_Name, (BYTE *) pAddr)

Command_Name: The SMBus command name that the User Module will use to map the command name to its RAM buffer. Possible values for this parameter are: WRITE_BYTE, WRITE_WORD, READ_BYTE, READ_WORD, BLOCK_WRITE, and BLOCK_READ.

pAddr: A pointer to the RAM buffer for the given command name.

Exceptions:

- The Quick Command, Send Byte and Receive Byte commands do not need to be assigned a RAM Buffer. These commands use default single byte RAM variables.
- The Process Call command uses the Write Word and Read Word command buffers.
- The Block Write Block Read Process Call command uses the Block Write and Block Read command buffers.

Buffer lengths:

- Write Byte and Read Byte command buffers = one byte
- Write Word and Read Word command buffers = two bytes
- Block Write = Block Write Buffer Length parameter

  Block Ready = Block Read Buffer Length parameter

  Note The sum of these two buffers cannot exceed 32 bytes

Refer to the Sample Firmware Source Code for the initialization code.

Command Descriptions

Quick Command

The SMBus Slave sets/clears a pre-defined flag (variable) which reflects the R/W bit received through this command.

Send Byte

The SMBus Slave writes the received byte to a pre-defined user module variable.

Receive Byte

The SMBus Slave sends the byte variable from a pre-defined user module variable.

Write Byte

When the SMBus Slave receives a byte through this protocol along with a valid command code defined by the user, it writes the byte to the RAM buffer initialized by the user.

Write Word

When the SMBus Slave receives a word through this protocol, along with a valid command code defined by the user, it writes the word to the RAM buffer initialized by user.

Read Byte

When the SMBus Slave receives this command along with a valid command code defined by the user, it sends one byte from the RAM buffer initialized by user.

Read Word

When the SMBus Slave receives this command along with a valid command code defined by the user, it sends the word from the RAM buffer initialized by user.

Process Call

When the SMBus Slave receives this command along with a valid command code defined by the user, it receives the word sent by the master, writes it to the RAM buffer assigned to the Write Word command, and sends the word from the RAM buffer assigned to the Read Word command.

Block Write

When the SMBus Slave receives this command along with a valid command code defined by the user, it checks the Byte Count (N) the master wants to write. If N is less than or equal to the Block Write Buffer Length parameter, the slave reads N bytes from the master, and writes them to the RAM buffer assigned to the Block Write command. If N is greater than the Block Write Buffer Length parameter, it NACKs the Byte Count byte sent by the master.

Block Read

When the SMBus Slave receives this command along with a valid command code defined by the user, it sends the Byte Count (N) to the master, where N is equal to the Block Read Buffer Length parameter. Then, it sends N bytes to the master from the RAM buffer assigned to the Block Read command.

Block Write-Block Read Process Call

When the SMBus Slave receives this command along with a valid command code defined by the user, it checks the Byte Count (N) the master wants to write. If N is less than or equal to the Block Write Buffer Length parameter, it reads N bytes from the master, and writes them to the RAM buffer assigned to the Block Write command. After a repeated start, the SMBus Slave sends the Byte Count (M) to the master, where M is equal to the Block Read Buffer Length parameter. Then, it sends M bytes to the master from the RAM assigned to the Block Read command. The total bytes transferred during this command must be less than 32 bytes.

SMBus Host Notify Protocol

This protocol will not be initiated by the master. Whenever the user wants to notify the host, they can do so using SMBusSlave_NotifyHost().

SMBusSlave_NotifyHost (BYTE Type, BYTE Databyte1, BYTE Databyte2)

Type: This determines the method for notifying the host:

0 = SMBALERT pin method

1 = Host notify command method

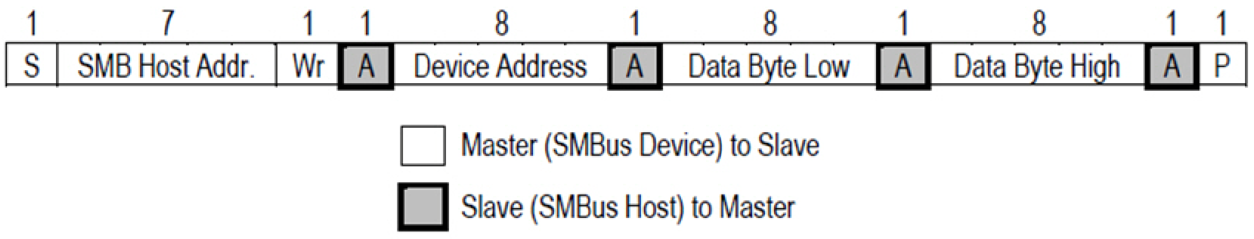
Databyte1: This is the Data Byte Low of the SMBus Host Notify protocol

Databyte2: This is Data Byte High byte of the SMBus Host Notify protocol

Note The Databyte1 and Databyte2 parameters are valid only for the HNP method. For the SMBALERT pin method, these parameters are ignored.

The user module will momentarily become the master and drive both SMBDAT and SMBCLK. The SMBus Slave will use the byte sequence shown in Figure 2 to notify the host.

Figure 2. Host Notify Protocol



DC and AC Electrical Characteristics

The following values are indicative of expected performance and based on initial characterization data. Unless otherwise specified, temperature ranges are –40 °C to 125 °C, supply voltages range from 2.95 V to 5.3 V.

Table 1. DC Electrical Characteristics

Symbol	Parameter	Limits		Units
		Min	Max	
VIL	Data, Clock Input Low Voltage	–	0.8	V
VIH	Data, Clock Input High Voltage	2.1	VDD	V
VOL	Data, Clock Output Low Voltage	–	0.4	V
ILEAK	Input leakage	–	±5	µA
IPULLUP	Current through pull-up resistor or current source	100	350	µA
VDD	Nominal bus voltage	2.7	5.5	V

Note The $T_{TIMEOUT}$, T_{SEXT}, and T_{MEXT} specifications mentioned in SMBus Specification Version 2.0, are not met by this user module because of the hardware limitation.

Table 2. AC Electrical Characteristics

Symbol	Parameter	Limits		Units
		Min	Max	
FSMB	SMBus Operating Frequency	10	100	kHz
TBUF	Bus free time between Stop and Start Condition	4.7	–	µs
THD:STA	Hold time after (Repeated) Start Condition. After this period, the first clock is generated	4.0	–	µs
TSU:STA	Repeated Start Condition setup time	4.7	–	µs
TSU:STO	Stop Condition setup time	4.0	–	µs
THD:DAT	Data hold time	300	–	ns
TSU:DAT	Data setup time	250	–	ns
TLOW	Clock low period	4.7	–	µs
THIGH	Clock high period	4.0	–	µs
TF	Clock/Data Fall Time	–	300	ns
TR	Clock/Data Rise Time	–	1000	ns
TPOR	Time in which a device must be operational after power-on reset	–	500	ms

Placement

The SMBus Slave User Module does not require any digital or analog PSoC blocks. It consumes one I^2C controller block and its dedicated interrupt. The devices that have multiple I^2C controller blocks will support multiple SMBus Slave User Module placements. Multiple placements are not supported in other devices.

Parameters and Resources

After a SMBus Slave User Module is selected and placed using the Device Editor, values may be selected and altered for the following parameters.

Slave Address

This parameter selects the 7-bit slave address for the SMBus_Slave User Module.

Type	Char
Range	0–127
Default	0

Dependence on other parameters:

- If the Hardwired Slave Address is enabled, the slave address is limited to only the most significant five bits. The remaining two bits are set to Slave Address Pin A0 and Slave Address Pin A1.
- If the Hardwired Slave Address is disabled, the slave address is all seven bits of this parameter.

Hardwired Slave Address

This parameter selects whether the two least significant bits of the slave address are hardwired.

Type	Boolean
Range	Enable/Disable
Default	Disable

Packet Error Check

This parameter checks whether to use PEC during communication.

Type	Boolean
Range	Enable/Disable
Default	Enable

Auto Address Check

This parameter selects whether the hardware address recognizing feature is enabled. If it is disabled, the hardware address comparison feature is not available. This parameter is available only in the CY8C28xxx family PSoC devices.

Type	Boolean
Range	Enable/Disable
Default	Enable

SMBus Clock

This parameter selects the clock speed used with the SMBus slave.

Type	Integer
Range	50 kHz–100 kHz
Default	100 kHz

SMBus Pins

This parameter selects the Port 1 pins for the SMBus Slave signals (SMBDAT and SMBCLK).

Type	Boolean
Range	P1[0] - P1[1] or P1[5] - P1[7]
Default	P1[0] - P1[1]

SMBALERT Pin

This parameter selects the pin for SMBALERT signal.

Type	Enum
Range	Any GPIO
Default	None

Slave Address Pin A0

This parameter selects the pin for A0 bit of slave address.

Type	Enum
Range	Any GPIO
Default	None

Dependence on other parameters:

- If the Hardwired Slave Address is enabled, this parameter is valid
- If the Hardwired Slave Address is disabled, this parameter is grayed out

Slave Address Pin A1

This parameter selects the pin for A1 bit of slave address.

Type	Enum
Range	Any GPIO
Default	None

Dependence on other parameters:

- If the Hardwired Slave Address is enabled, this parameter is valid
- If the Hardwired Slave Address is disabled, this parameter is grayed out

Block Write Buffer Length

This parameter sets the RAM buffer length for the Block Write command.

Type	Integer
Range	1–32
Default	1

Block Read Buffer Length

This parameter sets the RAM buffer length for the Block Read command.

Type	Integer
Range	1–32
Default	1

Application Programming Interface

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

Note

In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X 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.

Slave Functions

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

SMBusSlave_Start

Description:

Enables the SMBusSlave.

C Prototype:

```
void   SMBusSlave_Start (void);
```

Assembly:

```
lcall  SMBusSlave_Start
```

Parameters:

None

Return Value:

None

SMBusSlave_SetAddr

Description:

This function sets the slave address that is recognized by the SMBus Master. The range of this address should be limited depending on the Hardwired Slave Address parameter.

C Prototype:

```
void   SMBusSlave_SetAddr (BYTE bAddr);
```

Assembly:

```
movA, 0x05
lcall  SMBusSlave_SetAddr
```

Parameters:

BYTE bAddr: Slave address range depending on Hardwired Slave Address

Return Value:

None

SMBusSlave_Stop

Description:

Disables the SMBusSlave.

C Prototype:

```
void SMBusSlave_Stop (void);
```

Assembly:

```
lcall SMBusSlave_Stop
```

Parameters:

None

Return Value:

None

SMBusSlave_SetBuffer

Description:

Configures the RAM Buffer for reading and writing operations from the master for a particular command.

C Prototype:

```
void SMBusSlave_SetBuffer (BYTE bCommand_Name, BYTE * pAddr);
```

Assembly:

```
lcall SMBusSlave_SetBuffer
```

Parameters:

BYTE Command_Name: The SMBus command being initialized.

BYTE Command_Code: The command code for the specified command name.

Return Value:

None

SMBusSlave_SetCommand

Description:

Assigns a command code for a specific SMBus command.

C Prototype:

```
void SMBusSlave_SetCommand (BYTE Command_Name, BYTE Command_Code);
```

Assembly:

```
lcall SMBusSlave_SetCommand
```

Parameters:

BYTE Command_Name: The SMBus command being initialized.

BYTE Command_Code: The command code for the specified command name.

Return Value:

None

SMBusSlave_NotifyHost

Description:

Notifies the host either through the SMBALERT pin or through the Host notify command depending on the "Type" parameter.

C Prototype:

```
void SMBusSlave_NotifyHost (BYTE Type, BYTE Databyte1, BYTE Databyte2);
```

Assembly:

```
lcall SMBusSlave_NotifyHost
```

Parameters:

BYTE Type: This determines the method of notifying the host.

0 = SMBALERT pin

1 = Host notify command

BYTE Databyte1: This is the data byte 1 of the SMBus Host notify protocol. This parameter is only valid when Type = 1.

BYTE Databyte2: This is the data byte 2 of the SMBus Host notify protocol. This parameter is only valid when Type = 1.

Return Value:

None

Sample Firmware Source Code

The following C code illustrates the use of the APIs:

```
//------------------------------------------------------------------------------
// SMBusSlave sample code
// UM should be configured as follows:
// - Name: SMBusSlave
// - Slave Address: 4
// - Block Write Buffer Length: 6
// - Block Read Buffer Length: 6
// All other UM parameters should be left by default
//
//Here is almost SMBus Protocols to use with Bridge Control Panel tool:
// w 4 bb 80 p ;send byte NOTIFYHOST (PEC 80)
// r 4 x x p  ;receive byte (PEC e2)
// w 4 40 b6 01 p ;write byte SMBALERT (PEC 01)
// w 4 50 ab cd 76 p ;write word (PEC 76)
// w 4 60 r 4 x x p ;read byte (PEC 82)
// w 4 70 r 4 x x x p ;read word (PEC b6)
// w 4 80 ab cd r 4 x x x p ;process call (PEC 83)
// w 4 20 4 1 2 3 4 bd p ;block write (PEC bd)
// w 4 30 r 4 x x x x x x x x p ;block read (PEC a6)
// w 4 10 5 2 3 4 5 6 r 4 x x x x x x x x p ;block process call (PEC 04)
//------------------------------------------------------------------------------

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

BYTE baBlockWriteBuffer[SMBusSlave_BLOCK_WRITE_BUFFER_LENGTH];
BYTE baBlockReadBuffer[] = {10, 11, 12, 13, 14, 15};
BYTE WriteByteBuffer;
WORD WriteWordBuffer;
BYTE ReadByteBuffer = 0xad;
WORD ReadWordBuffer = 0xbcde;

void main(void)
{
    M8C_EnableGInt;
    SMBusSlave_Start();

    SMBusSlave_SetCommand(SMBusSlave_BLOCK_WRITE, 0x20);
    SMBusSlave_SetCommand(SMBusSlave_BLOCK_READ, 0x30);
    SMBusSlave_SetCommand(SMBusSlave_WRITE_BYTE, 0x40);
    SMBusSlave_SetCommand(SMBusSlave_WRITE_WORD, 0x50);
    SMBusSlave_SetCommand(SMBusSlave_READ_BYTE, 0x60);
    SMBusSlave_SetCommand(SMBusSlave_READ_WORD, 0x70);
    SMBusSlave_SetCommand(SMBusSlave_PROCESS_CALL, 0x80);
    SMBusSlave_SetCommand(SMBusSlave_BLOCK_WRITE_BLOCK_READ_PROCESS_CALL, 0x10);

    SMBusSlave_SetBuffer(SMBusSlave_BLOCK_WRITE, baBlockWriteBuffer);
    SMBusSlave_SetBuffer(SMBusSlave_BLOCK_READ, baBlockReadBuffer);
    SMBusSlave_SetBuffer(SMBusSlave_WRITE_BYTE, &WriteByteBuffer);
    SMBusSlave_SetBuffer(SMBusSlave_WRITE_WORD, (BYTE *) &WriteWordBuffer);
    SMBusSlave_SetBuffer(SMBusSlave_READ_BYTE,  &ReadByteBuffer);
    SMBusSlave_SetBuffer(SMBusSlave_READ_WORD, (BYTE *) &ReadWordBuffer);
    SMBusSlave_bReceiveByte = 0xaa;

    while(1)
    {
        if(SMBusSlave_bSendByte == 0xbb)
        {
            SMBusSlave_bSendByte = 0;
            SMBusSlave_NotifyHost(SMBusSlave_NOTIFYHOST, 5, 6);
        }
        if(WriteByteBuffer == 0xb6)
        {
            WriteByteBuffer = 0;

            SMBusSlave_NotifyHost(SMBusSlave_SMBALERT, 5, 6);
        }
    }
}
```

The following is the Assembly sample code:

```
;-----------------------------------------------------------------
; SMBusSlave sample code
;
; UM should be configured as follows:
; - Name: SMBusSlave
; - Slave Address: 4
; - Block Write Buffer Length: 6
; - Block Read Buffer Length: 6
; All other UM parameters should be left by default
;
; Here are some SMBus Protocols to use with Bridge Control Panel tool:
; w 4 bb 80 p ;send byte NOTIFYHOST (PEC 80)
; w 4 40 b6 01 p ;write byte SMBALERT (PEC 01)
; w 4 20 4 1 2 3 4 bd p ;block write (PEC bd)
; w 4 30 r 4 x x x x x x x x p ;block read (PEC a6)
; w 4 10 5 2 3 4 5 6 r 4 x x x x x x x x p ;block process call (PEC 04)
;-----------------------------------------------------------------

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

export _main
export baBlockWriteBuffer
export baBlockReadBuffer
export bWriteByteBuffer

area data (rel,con,ram)

baBlockWriteBuffer: BLK SMBusSlave_BLOCK_WRITE_BUFFER_LENGTH
baBlockReadBuffer: BLK SMBusSlave_BLOCK_READ_BUFFER_LENGTH
bWriteByteBuffer: BLK 1

area text (rel,con,rom,code)
```

```
_main:
    mov [baBlockReadBuffer + 0], 10
    mov [baBlockReadBuffer + 1], 11
    mov [baBlockReadBuffer + 2], 12
    mov [baBlockReadBuffer + 3], 13
    mov [baBlockReadBuffer + 4], 14
    mov [baBlockReadBuffer + 5], 15
    mov [SMBusSlave_bReceiveByte], aah

    M8C_EnableGInt
    lcall SMBusSlave_Start

    mov X, 40h
    mov A, SMBusSlave_WRITE_BYTE
    lcall SMBusSlave_SetCommand
    mov X, 30h
    mov A, SMBusSlave_BLOCK_READ
    lcall SMBusSlave_SetCommand
    mov X, 20h
    mov A, SMBusSlave_BLOCK_WRITE
    lcall SMBusSlave_SetCommand
    mov X, 10h
    mov A, SMBusSlave_BLOCK_WRITE_BLOCK_READ_PROCESS_CALL
    lcall SMBusSlave_SetCommand

    mov A, >baBlockReadBuffer
    push A
    mov A, <baBlockReadBuffer
    push A
    mov A, SMBusSlave_BLOCK_READ
    push A
    lcall SMBusSlave_SetBuffer
    add SP, -3

    mov A, >baBlockWriteBuffer
    push A
    mov A, <baBlockWriteBuffer
    push A
    mov A, SMBusSlave_BLOCK_WRITE
    push A
    lcall SMBusSlave_SetBuffer
    add SP, -3

    mov A, >bWriteByteBuffer
    push A
    mov A, <bWriteByteBuffer
    push A
    mov A, SMBusSlave_WRITE_BYTE
    push A
    lcall SMBusSlave_SetBuffer
    add SP, -3

.mainloop:
    cmp [SMBusSlave_bSendByte], bbh
    jnz .SmbAlertCheck
    mov [SMBusSlave_bSendByte], 0
    mov A, 6 ;Data Byte High
    push A
    mov A, 5 ;Data Byte Low
    push A
    mov A, SMBusSlave_NOTIFYHOST
    push A
    lcall SMBusSlave_NotifyHost
    add SP, -3

.SmbAlertCheck:
    cmp [bWriteByteBuffer], b6h
    jnz .mainloop
    mov [bWriteByteBuffer], 0
    mov A, SMBusSlave_SMBALERT
    push A
    lcall SMBusSlave_NotifyHost
    add SP, -1
    jmp .mainloop
```

Configuration Registers

This section describes the PSoC Resource Registers used or modified by the SMBusSlave User Module. The use of these registers is not required when using the SMBusSlave User Module, but is provided as a reference.

Table 3. 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]	0	Enable Slave

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

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

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

Clock Rate[1,0]: Selects among three valid clock rates 50, 100, or 400 kbps.

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

Table 4. Resource I2C_SCR: Bank 0 reg[D7] Status Control Register

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

Bus Error: Indicates detection of a Bus Error condition.

Stop Status: Indicated detection of an I^{2}C stop condition.

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

Address: Received or transmitted byte is an address.

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

Byte Complete: 8 data bits were received. For Receive Mode, the bus is stalled waiting for an ACK/NACK. For Transmit Mode ACK/NACK was also received (see LRB) and the bus is stalled for the next action.

Table 5. 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, you must load this register before a write to the I2C_SCR register. Received data is read from this register and may contain an address or data.

Table 6. Resource I2C_ADDR: Bank 1 reg[AD] Data Register

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Value	Data							

The I^{2}C address register is used to configure the hardware address automatic comparison feature so that the microcontroller is not disturbed by an unwanted slave request. The hardware address automatic compare feature is available in the Slave only mode in the CY8C21x45, CY8C22x45, and CY8C28xxx family PSoC devices.

Version History

Version	Originator	Description
1.00	DHA	Initial version.
2.00	MYKZ	1. Corrected method of clearing posted interrupts. 2. Fixed I2C address saving in User Module parameters and corrected I2C address constant generation.

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.