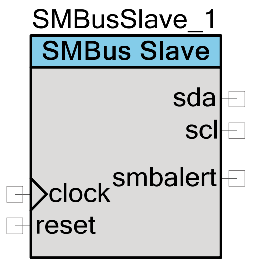
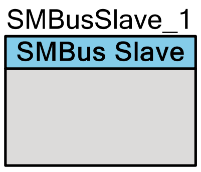
SMBus and PMBus Slave

3.0

PSoC 3/PSoC 5LP



PSoC 4



Features

- SMBus/PMBus Slave mode
- SMBALERT# pin support
- 25 ms Timeout
- Configurable SMBus/PMBus commands
- Packet Error Checking (PEC) support

General Description

The System Management Bus (SMBus) and Power Management Bus (PMBus) Slave component provides a simple way to add a well-known communications method to a PSoC 3-, PSoC 4-, or PSoC 5LP-based design.

SMBus is a two-wire interface that is often used to interconnect a variety of system management chips to one or more host systems. It uses I^2C with some extensions as the physical layer. There is also a protocol layer, which defines classes of data and how that data is structured. Both the physical layer and protocol layer add a level of robustness not originally embodied in the I^2C specification. The SMBus Slave component supports most of the SMBus version 2.0 Slave device specifications with numerous configurable options.

PMBus is an extension to the more generic SMBus protocol with specific focus on power conversion and power management systems. With some slight modifications to the SMBus protocol, the PMBus specifies application layer commands, which are not defined in the SMBus. The PMBus component presents all possible PMBus revision 1.2 commands and allows you to select which commands are relevant to an application.

When to Use an SMBus and PMBus Slave

This component can be used in a design that requires an SMBus or PMBus slave communication interface. The component handles most of the physical layer requirements in the hardware. The rest of the work, including the protocol and memory management, is handled in firmware.

Input/Output Connections

This section describes the various input and output connections of the SMBus and PMBus Slave. An asterisk (*) in the list of I/Os indicates that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

clock – Input *

This is the clock that operates this block. On PSoC 3 and PSoC 5LP, this terminal is displayed when the I^{2}C Implementation parameter is set to Universal Digital Block (UDB) and the UDB clock source parameter is set to External clock. The clock provided must be 16 times the desired data rate as shown in the following table.

Data Rate	Clock
10 kbps	160 kHz
50 kbps	800 kHz
100 kbps	1.6 MHz
400 kbps [1]	6.4 MHz

For PSoC 4, the terminal is visible if the Clock from terminal parameter is selected on **I^{2}C Configuration** tab. The input clock and Oversampling factor parameter determines the actual data rate.

Note When setting the data rate or external clock frequency value, make sure that PSoC Creator can provide this value using the current system clock frequency. Otherwise, a warning about the clock accuracy range is generated while building the project. This warning contains the actual clock value set by PSoC Creator. Choose whether the system clock or component clock should be changed to fit the clocking system requirements and achieve an optimal value.

reset – Input *

This is the hardware reset for the UDB I^{2}C implementation, and it applies to PSoC 3 and PSoC 5LP only. If the active-high reset pin is held to logic high, the I^{2}C block is held in reset and communication over I^{2}C stops. SDA and SCL are forced to high. This is a hardware reset only. Software must be independently reset using the Stop() and Start() APIs. The reset input may be left floating with no external connection. If nothing is connected to the reset line, the component will assign it a constant logic 0.

1. PMBus mode only.

sda (SMBDAT) – Input/Output *

This is the serial data input/output. It is a bidirectional data signal used to transmit or receive all bus data. The pin connected to sda should be configured to have the **Drive Mode** parameter set to “Open Drain, Drives Low”.

If you select the External IO buffer option on this component, the single sda bidirectional signal is replaced by a separate input and output (sda_o and sda_i). This is necessary to enable the multiplexing of multiple I^2C buses required by certain applications.

Note The pin is buried inside the component on PSoC 4. The External IO buffer option is not supported on PSoC 4.

scl (SMBCLK) – Input/Output *

This is the serial clock input/output. Although the slave never generates the clock signal, it may hold the clock low, stalling the bus until it is ready to send data or ACK/NACK the latest data or address. The pin connected to scl should be configured to have **Drive Mode** parameter set to “Open Drain, Drives Low”.

If you select the External IO buffer option on this component, the single scl bidirectional signal is replaced by a separate input and output (scl_o and scl_i). This is necessary to enable the multiplexing of multiple I^2C busses required by certain applications.

Note The pin is buried inside the component on PSoC 4. The External IO buffer option is not supported on PSoC 4.

smbalert (SMBALERT#) – Output *

This is an optional SMBus alert signal. It displays when the Enable SMBALERT# pin option is checked. The alert pin may be asserted/de-asserted via APIs described later. The pin connected to smbalert should be configured as “Open Drain, Drives Low”.

Note The pin is buried inside the component on PSoC 4.

scl_timeout – Input *

This is the serial clock (scl) low timeout detection.

The PSoC 4 implementation requires a dedicated device pin for scl low timeout detection. The pin must be connected to the SMBCLK line of the bus external to the PSoC device.

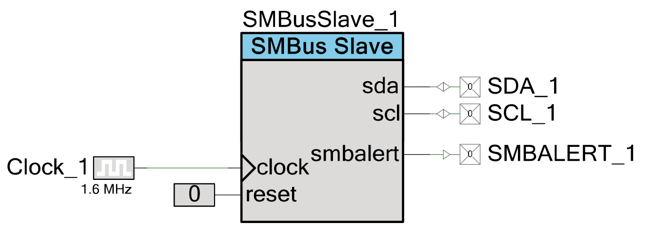
On PSoC 3 and PSoC 5LP, the component monitors the scl signal internally and does not use this input signal.

Schematic Macro Information

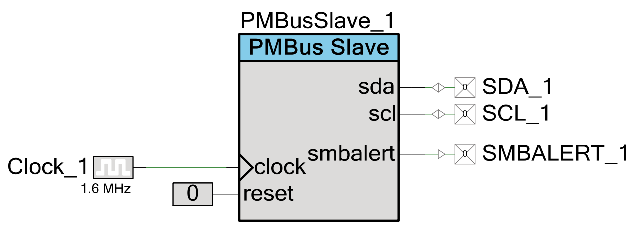
For PSoC 3 and PSoC 5LP, the PSoC Creator Component Catalog contains two schematic macro implementations. These macros contain either the SMBus Slave or PMBus Slave component, already connected to digital pins and a clock. The components are configured in SMBus Slave or PMBus Slave mode respectively, and use UDB-based I^{2}C implementations.

The pins connected to scl, sda and smbalert terminals are configured with the **Drive Mode** parameter set to "Open Drain, Drives Low". The clock frequency is set for 100 kbps data rate.

SMBus Slave Macro



PMBus Slave Macro

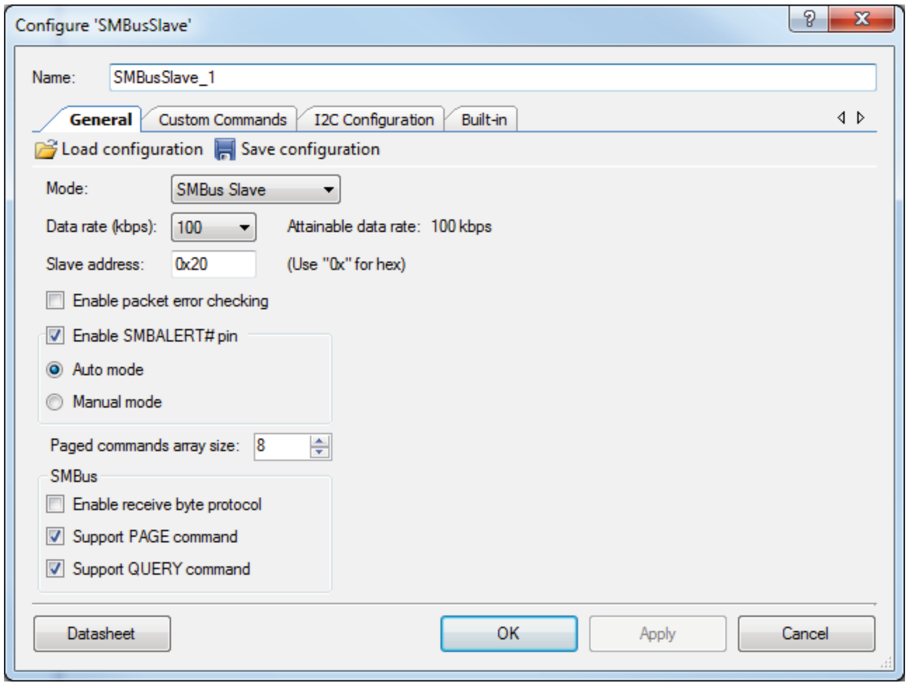


Component Parameters

Drag an SMBus/PMBus Slave component onto your design and double click it to open the Configure dialog. This dialog has the following tabs with different parameters.

General Tab

The **General** tab provides options to configure the general settings of the SMBus/PMBus.



Name: SMBusSlave_1

Mode: SMBus Slave

Data rate (kbps): 100 Attainable data rate: 100 kbps

Slave address: 0x20 (Use "0x" for hex)

☐ Enable packet error checking

☑ Enable SMBALERT# pin

● Auto mode

○ Manual mode

Paged commands array size: 8

SMBus

☐ Enable receive byte protocol

☑ Support PAGE command

☑ Support QUERY command

Load configuration

This button restores all settings, including tables, from an external file.
Keyboard shortcut – [Ctrl] [L].

Save configuration

This button saves all settings, including tables, in an external file. Keyboard shortcut – [Ctrl] [S].

Mode

This parameter specifies the mode in which the component operates: SMBus Slave (default) or PMBus Slave. This parameter also determines the available data rate.

- In PMBus Slave mode there are two options: 100 and 400 kbps.
- In SMBus Slave mode there are additional 10 kbps and 50 kbps options.

Data rate

This parameter specifies the data transfer rate. The available options are dependent on the SMBus/PMBus Mode selection. The PMBus Slave mode allows data rate values of 100 kbps and 400 kbps. The SMBus Slave mode provides options for 10kbps, 50 kbps, 100 kbps, and 400 kbps. The default setting is 100 kbps.

Slave address

This parameter determines the primary address (7-bit format) of the device. The value can be entered either in decimal or in hexadecimal (if preceded by "0x"). The customizer validates the address to ensure that it does not conflict with any of the SMBus Slave reserved addresses. (Default = 0x20).

Enable packet error checking

This parameter enables/disables support for SMBus packet error checking (PEC).

- If selected, the component will perform the slave transfer with or without PEC, verify the correctness of the PEC if present, and only process the message if PEC is correct. If the received and calculated PEC bytes do not match, the component will respond as described in Corrupted Data section later in this document.
- If not selected (default), it is the host responsibility to know that PEC is not supported and to not send a PEC byte.

Enable SMBALERT# pin

This parameter allows you to configure the optional SMBALERT# output pin for host notification. If this option is selected, the pin becomes visible as an output on the component symbol.

The **Auto mode/Manual mode** options determine whether the SMBALERT# pin will automatically de-assert after the host queries the device at the Alert Response Address. See Appendix A of the SMBus specification for details. The SMBALERT# pin is enabled and set to **Auto mode** by default. The option can be changed by the SetSmbAlertMode() API call.

In **Auto mode**, SMBALERT# is automatically de-asserted once the bus master successfully reads the Alert Response Address performing a modified Receive Byte operation. It is "modified" because the host accesses the Alert Response Address and the device responds with its primary address; refer to the SMBus specification for more information.

In **Manual mode**, the component will call the HandleSmbAlertResponse() API, where user code (in a merge section) is responsible for de-asserting SMBALERT#.

Note The **Auto mode** option is not recommended for multi-slave applications. The component has no arbitration capability to determine that the highest priority (lowest address) device wins communication rights in the case where more than one device that pulled SMBALERT# low responds to the Alert Response address. This case results in de-asserting SMBALERT# by all devices that pulled the pin low. The preferred method in this case is to handle SMBALERT# in an application layer and de-assert the pin calling SetSmbAlert() API after a host reads the device fault status for example.

The arbitration on the I^2C bus relies on the wired AND connection of all devices to the I^2C bus. If more than one device tries to put information onto the bus, zero is transmitted if at least one device sends zero onto the bus. This constrains the slave address selection for the application. That is, the lowest address should not have ones in the address bits where higher addresses have zeros. For example, if two 7-bit slave addresses are 0100000 and 1000000, and both devices try to put their addresses onto the bus, the value 0000000 will be received by the host.

Paged commands array size

Determines the array size for paged commands. All paged commands share this array size. Range=1-64. (Default=8).

SMBus Box

The SMBus box configures the optional SMBus features. It is present only in SMBus Slave Mode.

- The **Enable receive byte protocol** check box enables/disables support for the SMBus Receive Byte protocol defined in the SMBus specification. If unchecked, any Receive Byte transaction is treated as a bus error. If checked, the component calls the GetReceiveByteResponse() API to determine the response byte for Receive Byte requests.

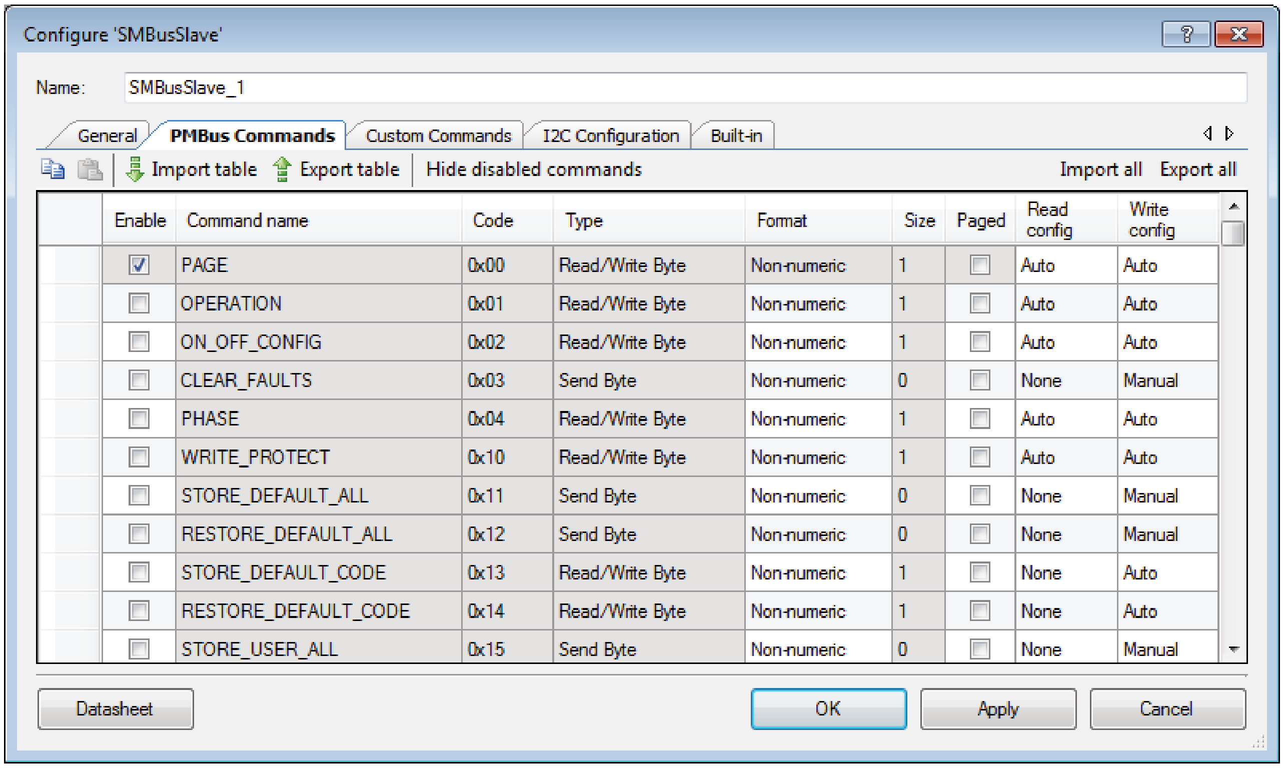
- The **Support PAGE Command** check box allows you to have access to the PAGE command while in **SMBus Slave** mode.
- The **Support QUERY Command** checkbox allows you to have access to the QUERY command while in **SMBus Slave** mode.

If either the PAGE or QUERY commands are enabled, then these commands are added to the command list on the Custom Commands tab. The properties for these commands are based on the PMBus specification, but a customization of the command codes is also possible.

The default values for this box are: Enable receives byte protocol unchecked, Support PAGE Command enabled, Command Code=0x00, Support QUERY command enabled, Command Code=0x1A.

PMBus Commands Tab

This tab is available when the **PMBus Slave** option is selected for the Mode parameter. It presents the entire list of defined commands from the PMBus specification. The pre-filled information includes the command name, numeric code, and command type. You may select the commands to be supported by selecting/unselecting an **Enable** check box associated with each command. **Command name**, **Code**, and **Type** are read-only fields.



Name: SMBusSlave_1

Enable	Command name	Code	Type	Format	Size	Paged	Read config	Write config
✓	PAGE	0x00	Read/Write Byte	Non-numeric	1	☐	Auto	Auto
☐	OPERATION	0x01	Read/Write Byte	Non-numeric	1	☐	Auto	Auto
☐	ON_OFF_CONFIG	0x02	Read/Write Byte	Non-numeric	1	☐	Auto	Auto
☐	CLEAR_FAULTS	0x03	Send Byte	Non-numeric	0	☐	None	Manual
☐	PHASE	0x04	Read/Write Byte	Non-numeric	1	☐	Auto	Auto
☐	WRITE_PROTECT	0x10	Read/Write Byte	Non-numeric	1	☐	Auto	Auto
☐	STORE_DEFAULT_ALL	0x11	Send Byte	Non-numeric	0	☐	None	Manual
☐	RESTORE_DEFAULT_ALL	0x12	Send Byte	Non-numeric	0	☐	None	Manual
☐	STORE_DEFAULT_CODE	0x13	Read/Write Byte	Non-numeric	1	☐	None	Auto
☐	RESTORE_DEFAULT_CODE	0x14	Read/Write Byte	Non-numeric	1	☐	None	Auto
☐	STORE_USER_ALL	0x15	Send Byte	Non-numeric	0	☐	None	Manual

Toolbar

- **Copy/Paste –** These commands allow you to copy selected table rows and paste them to the **Custom Commands** tab.

  Note The Paste function cannot be used on any other tab.

- **Import/Export table –** These options allow you to save and restore the settings to/from an external file. This allows for easy loading of preset profiles and retention of custom settings.
- **Hide disabled commands –** This option allows you to hide commands that are not enabled.
- **Import all/Export all –** These commands allow you to save and restore the settings for both **PMBus Commands** and **Custom Commands** tables to an external file.

Format

This parameter specifies the data format for this command. This format is used by the component in formulating the response to the QUERY command. The possible format values available in the QUERY command are Non-numeric, Linear, Signed, Direct, Unsigned, VID Mode and Manufacturer. This field is only used for purposes of the QUERY command, as the component does not perform any actual numeric conversion. The default setting is Non-numeric.

Size

For Block and Process Call type commands, you may edit the **Size** field to specify the size of the data element. This size does not include the size/count byte that the SMBus protocol appends to the beginning of block transfers. This field can only be edited for Block or Process Call commands. For all other types, the **Size** field comes from the PMBus specification. The default value is 16.

Paged

This parameter indicates whether this command is **Paged** (i.e., indexed) or not. For commands that are paged, the component automatically generates an array for that command in the register store. The size of the array is determined by the Paged command's array size parameter. For Auto Read/Write config, the component automatically indexes to the correct array element of a **Paged** parameter based on the current PMBus page (as selected by the last Page command).

Read/Write config

For each command, select whether that command is readable and/or writeable via the **Read config** and **Write config** parameters. For each, select None, Auto, or Manual (default).

- None indicates the action is disabled (that is, set **Write config** to None for read-only).

- Auto indicates that commands are handled entirely by the component without user firmware intervention or notification. In the case of Auto write, the component automatically copies the command data from the transfer buffer to the appropriate variable in the Operating register store for the command. In the case of Auto read, the component automatically copies the data for the command from the Operating register store to the transfer buffer and completes the transaction.
- Manual indicates that commands must be handled by the user main program context out of the component ISR context. When a manual command arrives, the component will add it to the transaction queue and disable the interrupt. The component provides the following APIs to process the manual commands from the main program: GetTransactionCount(), GetNextTransaction() and CompleteTransaction(). Refer to the Application Programming Interface section for a detailed description of each API.

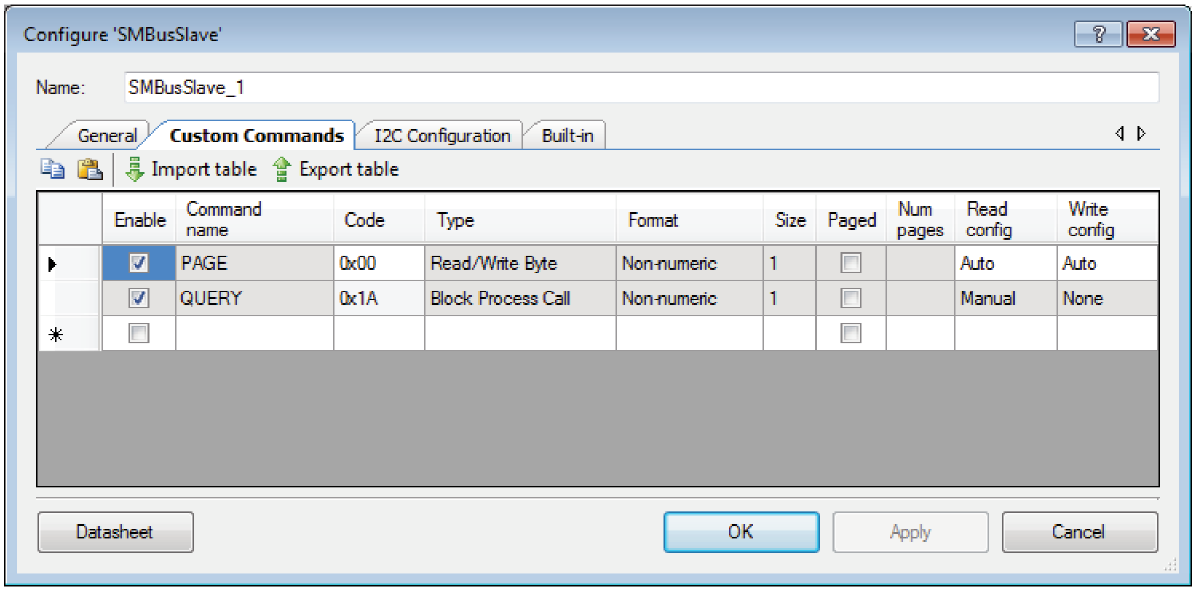
Note Because of the possible asymmetry between writes and reads and the complex nature of Process Call protocol, Auto may not be selected for commands that use that protocol.

It is more likely to enable Auto mode for a read than a write. The reason for this is that reads are less likely to require additional actions outside of the component, where writes often will do. For example, the PMBus over-voltage threshold command (VOUT_OV_FAULT_THRESHOLD) would have **Read config** set to Auto and **Write config** set to Manual in most systems.

User firmware would need to take action on the write of this parameter to set up hardware for the new threshold. However, if the PMBus host just wanted to read the currently configured value there is no need for user firmware to get involved. An example of a Read command that might not be configured in Auto mode would be the PMBus command READ_VOUT, which returns the measured voltage of an output rail. User firmware would need to take a voltage measurement, perform the unit translation and Operating register store update.

Custom Commands Tab

This tab allows you to enter all SMBus commands and any manufacturer-specific PMBus commands.



	Enable	Command name	Code	Type	Format	Size	Paged	Num pages	Read config	Write config
▶	✓	PAGE	0x00	Read/Write Byte	Non-numeric	1	☐		Auto	Auto
	✓	QUERY	0x1A	Block Process Call	Non-numeric	1	☐		Manual	None
*	☐						☐			

Toolbar

- **Copy/Paste –** These commands allow you to copy selected table rows and paste them within the **Custom Commands** tab.

  Note The Paste function cannot be used on any other tab.

- **Import/Export table –** These options allow you to save and restore settings to an external file. This allows for easy loading of preset profiles and retention of custom settings.

Command name

The user specified name for the command. Allowed characters are A-Z (all caps), 0-9, and the underscore "_". The maximum length is 24 characters. The first character may not be a number. Command name duplicates are not allowed (including custom command names that duplicate standard PMBus command names). The Command Name is blank by default.

Command code

The numeric code for this command. It is a hexadecimal value, limited to two characters. Duplicate command codes are not allowed. This includes command codes that conflict with enabled PMBus commands. The Command Code is left blank by default.

Type

This parameter specifies the SMBus transfer protocol/data-size used for this command. The possible values are Send Byte, Read/Write Byte, Read/Write Word, Read/Write Block, Process Call, and Block Process Call. It is set to Read/Write Byte as default.

Format

This parameter specifies the numeric format for this command. This format is used by the component in formulating the response to the QUERY command. The possible format values available in the QUERY command are Non-numeric, Linear, Signed, Direct, Unsigned, VID Mode and Manufacturer. This field is only used for purposes of the QUERY command, as the component does not perform any actual numeric conversion. The default setting is Non-numeric.

Size

For Block and Process Call type commands, you may edit the **Size** field to specify the size of the data element. This size does not include the size/count byte that the SMBus protocol appends to the beginning of block transfers. This field can only be edited for Block or Process Call commands. For all other types, the **Size** field comes from the PMBus Specification. The default value is 16.

Paged

This parameter indicates whether this command is paged (i.e., indexed) or not. For commands that are paged, the component automatically generates an array for that command in the register store. The size of the array is determined by the **Paged** command's array size parameter. For Auto Read/Write config, the component automatically indexes to the correct array element of a **Paged** parameter based on the current PMBus page (as selected by the last Page command).

Num pages

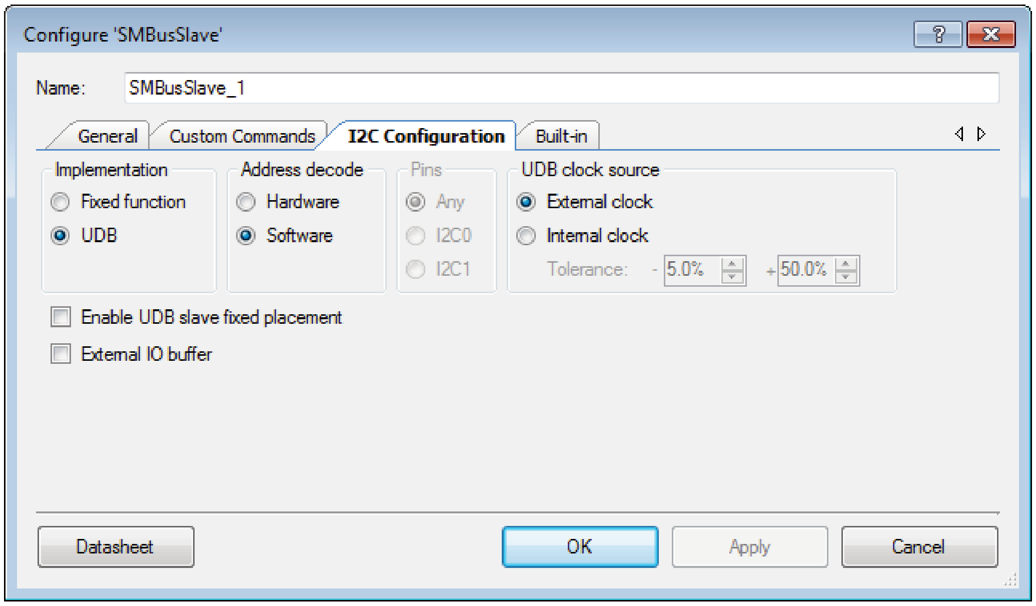
The number of pages parameter will default to the total number of pages specified on the **General** tab. That is, when you add a new command, the number of pages for that command will be equal to the Paged commands array size value on the **General** tab. However changing the value of the Paged commands array size parameter will not change the number of pages set for already entered custom commands. You may elect to reduce this parameter as it applies to this particular command. Minimum setting is 1. Maximum setting is the total number of pages selected on the **General** tab. This parameter is grayed out when the Paged check box is un-checked.

Read/Write Config

For each command, select whether that command is readable and/or writeable via the Read Config and Write Config parameters. For each, select None, Auto, or Manual (default). Refer to Read/Write config parameter description on **PMBus Commands** tab for details.

I²C Configuration Tab (PSoC 3 / PSoC 5LP)

This tab allows you to configure the I²C hardware.



Configure 'SMBusSlave'

Name: SMBusSlave_1

General | Custom Commands | **I2C Configuration** | Built-in

Implementation: Fixed function [✓ UDB]

Address decode: Hardware [✓ Software]

Pins: [✓ Any] I2C0 I2C1

UDB clock source: [✓ External clock] Internal clock

Tolerance: - 5.0% + 50.0%

☐ Enable UDB slave fixed placement

☐ External IO buffer

Implementation

This parameter determines whether the I²C hardware is implemented using Fixed Function or UDB. The default mode is set to UDB. Refer to the Resources section later in this document for details on device resources utilization.

Address decode

This parameter allows you to choose between software and hardware address decoding. For most applications where only one slave address is required, hardware address decoding is preferred. If hardware address decode is enabled, the block automatically NACKs addresses that are not its own without CPU intervention. It automatically interrupts the CPU on correct address reception, and holds the scl line low until CPU intervention.

Software address detection allows the component to respond to multiple addresses. That is, the software address detection must be used to support Alert Response Address and General Call Address. (Default = Software).

Pins

This parameter determines which type of pins to use for sda and scl signal connections. The setting is available for the Fixed function I²C implementation only. Options: Any, I2C0, and I2C1. (Default = Any). "Any" means general-purpose I/O (GPIO or SIO).

UDB clock source

This parameter allows you to choose between an internally configured clock and an externally configured clock for data rate generation. When set to Internal Clock, PSoC Creator calculates and configures the required clock frequency based on the Data rate parameter, taking into account 16 times oversampling. In External clock mode, the component does not control the data rate but displays the actual data rate based on the user-connected clock source.

If this parameter is set to Internal clock then the clock input is not visible on the symbol. You can enter the desired tolerance values for the internal clock. PSoC Creator will ensure that the accuracy of the resulting clock falls within the given tolerance range or produce a warning if the desired clock is not achievable. Clock tolerances are specified as a percentage. The valid range is -5% to +50%.

Enable UDB slave fixed placement

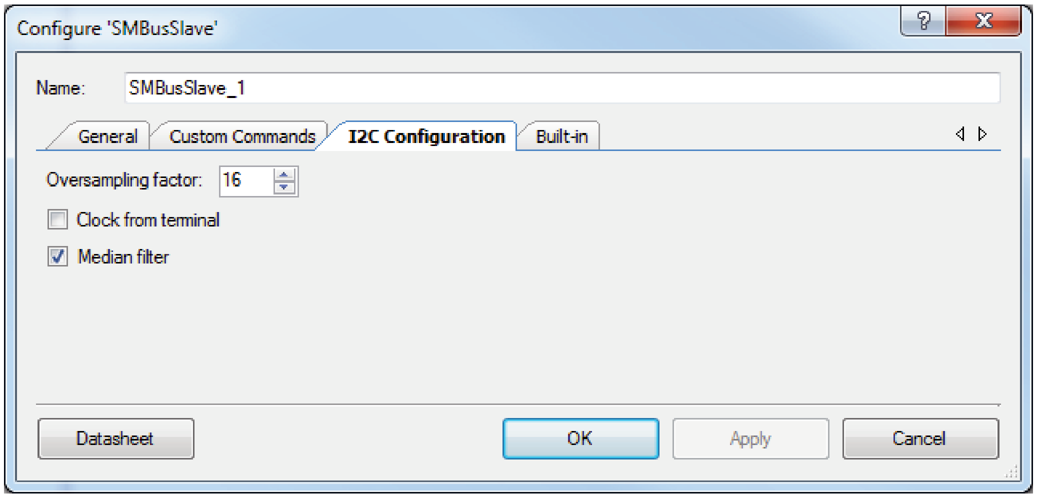
This parameter allows you to choose a fixed component placement that improves the component performance over unconstrained placement. If this parameter is set, all of the component resources are fixed in the top right corner of the device. This parameter controls the assignment of pins connected to the component. The choice of pin assignment is not a determining factor for component performance. This option is only valid if the Implementation parameter is set to UDB. The fixed placement aspect of the component removes the routing variability. It also allows the fixed placement to continue to operate the same as a non-fixed placed design would in a fairly empty design.

External IO buffer

This parameter allows internal I^{2}C bus multiplexing. The internal OE buffer is removed and bidirectional scl and sda terminals are replaced with separate inputs (sda_i and scl_i) and outputs (sda_o and scl_o). For more information on I^{2}C bus multiplexing implementation, refer to the PSoC Creator I^{2}C Component Datasheet.

I^{2}C Configuration Tab (PSoC 4)

This tab allows you to configure the I^{2}C hardware.



Name: SMBusSlave_1

Oversampling factor: 16

☐ Clock from terminal

☑ Median filter

Oversampling factor

This parameter defines the oversampling factor of the scl clock; the number of component clocks within one scl clock period. Oversampling factor is used to calculate the internal component clock frequency required to achieve this amount of oversampling for the defined Data rate. An oversampling factor maximum value is 32 and the minimum value depends on Median filter settings. The default is 16.

Clock from terminal

This parameter allows choosing between an internally configured clock and an externally configured clock for data rate generation. When the option is enabled, the component does not control the data rate, but displays the actual data rate based on the user-connected clock source and component oversampling factor. When this option is not enabled, PSoC Creator configures the required clock source. The clock source frequency is calculated by the component based on the Data rate parameter and Oversampling factor.

Median filter

This parameter applies a digital 3 tap median filter on the input path of sda. This filter reduces the susceptibility to errors. However, the minimum oversampling factor value is increased.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software at runtime. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "SMBusSlave_1" to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "SMBusSlave."

Note Some of the component API functions are used in component ISRs and, therefore, when building with the Keil compiler may provoke a compiler warning. To avoid this, include these functions in the ".cyre" file.

Function	Description
SMBusSlave_Start()	Initializes and enables the SMBus component. The I^2C interrupt is enabled, and the component can respond to the SMBus traffic.
SMBusSlave_Stop()	Stops responding to the SMBus traffic. Also disables the interrupt.
SMBusSlave_Init()	This function initializes or restores the component according to the customizer Configure dialog settings.
SMBusSlave_Enable()	Activates the hardware and begins component operation.
SMBusSlave_EnableInt()	Enables the component interrupt.
SMBusSlave_DisableInt()	Disables the interrupt.
SMBusSlave_SetAddress()	Sets the primary address.
SMBusSlave_SetAlertResponseAddress()	Sets the Alert Response Address.
SMBusSlave_SetSmbAlert()	Sets the value passed to the SMBALERT# pin.
SMBusSlave_SetSmbAlertMode()	Determines how the component responds to an SMBus master read at the Alert Response Address.
SMBusSlave_HandleSmbAlertResponse()	Called by the component when it responds to the Alert Response Address issued by the host and the SMBALERT Mode is set to MANUAL_MODE.
SMBusSlave_GetNextTransaction()	Returns a pointer to the next transaction record in the transaction queue. If the queue is empty, the function returns NULL.
SMBusSlave_GetTransactionCount()	Returns the number of transaction records in the transaction queue.
SMBusSlave_CompleteTransaction()	Causes the component to complete the currently pending transaction at the head of the queue.
SMBusSlave_GetReceiveByteResponse()	Returns the byte to respond to a "Receive Byte" protocol request.
SMBusSlave_HandleBusError()	Called by the component whenever a bus protocol error occurs.

Function	Description
SMBusSlave_StoreUserAll()	Saves the Operating register store in RAM to the User register store in Flash.
SMBusSlave_RestoreUserAll()	Verifies the CRC field of the User register store and then copies the contents of the User register store to the Operating register store.
SMBusSlave_EraseUserAll()	Erase the User register store in Flash.
SMBusSlave_RestoreDefaultAll()	Verifies the signature field of the Default register store and then copies the contents of the Default register store to the Operating register store.
SMBusSlave_StoreComponentAll()	Update the parameters of other components in the system with the current PMBus settings.
SMBusSlave_RestoreComponentAll()	Updates the PMBus Operating register store with the current configuration parameters of other components in the system.
SMBusSlave_Lin11ToFloat()	Converts the argument "linear11" to floating point and returns it.
SMBusSlave_FloatToLin11()	Takes the argument "floatvar" (a floating point number) and converts it to a 16-bit LINEAR11 value (11-bit mantissa + 5-bit exponent), which it returns.
SMBusSlave_Lin16ToFloat()	Converts the argument "linear16" to floating point and returns it.
SMBusSlave_FloatToLin16()	Takes the argument "floatvar" (a floating point number) and converts it to a 16-bit LINEAR16 value (16-bit mantissa), which it returns.

Global Variables

Function	Description
SMBusSlave_initVar	The initVar variable is used to indicate initial configuration of this component. This variable is prepended with the component name, in this case, SMBusSlave. The SMBusSlave_initVar variable is initialized to zero and set to 1 the first time SMBusSlaveStart() is called. This allows for component initialization without reinitialization in all subsequent calls to the SMBusSlave Start() routine. If it is necessary to reinitialize the component, then the SMBusSlave_Init() function can be called before the SMBusSlave_Start() or SMBusSlave_Enable() function.
SMBusSlave_regs	Refer to Operating Register Store section of this document for a detailed description.
SMBusSlave_regsDefault	Refer to Default Register Store section of this document for a detailed description.
SMBusSlave_transactionData[]	Transaction queue structure. Refer to Transaction Queue section of this document for a detailed description.

void SMBusSlave_Start(void)

Description:	This is the preferred method to begin component operation. SMBusSlave_Start() calls the SMBusSlave_Init() function, and then calls the SMBusSlave_Enable() function.
Parameters:	None
Return Value:	None
Side Effects:	None

void SMBusSlave_Stop(void)

Description:	This function stops the component and disables the interrupt. It releases the bus if it was locked up by the device and sets it to the idle state.
Parameters:	None
Return Value:	None
Side Effects:	None

void SMBusSlave_Init(void)

Description:	This function initializes or restores the component according to the customizer Configure dialog settings. It is not necessary to call SMBusSlave_Init() because the SMBusSlave_Start() API calls this function, which is the preferred method to begin component operation.
Parameters:	None
Return Value:	None
Side Effects:	All registers will be set to values according to the customizer Configure dialog.

void SMBusSlave_Enable(void)

Description:	This function activates the hardware and calls EnableInt() to begin component operation. It is not necessary to call SMBusSlave_Enable() because the SMBusSlave_Start() API calls this function, which is the preferred method to begin component operation. If this API is called, SMBusSlave_Start() or SMBusSlave_Init() must be called first.
Parameters:	None
Return Value:	None
Side Effects:	None

void SMBusSlave_EnableInt(void)

Description: This function enables the component interrupt. It is not required to call this API to begin the component operation since it is called in SMBusSlave_Enable().

Parameters: None

Return Value: None

Side Effects: None

void SMBusSlave_DisableInt(void)

Description: This function disables the component interrupt. This function is not normally required because the I2C_Stop() function disables the interrupt. The component does not operate when the interrupt is disabled.

Parameters: None

Return Value: None

Side Effects: If the interrupt is disabled while the component is still running, it can cause the bus to lock up.

void SMBusSlave_SetAddress(uint8 address)

Description: This function sets the primary slave address of the device.

Parameters: uint8 address: primary device address. This value can be any address between 0 and 127 (0x00 to 0x7F) except reserved SMBus addresses. The address is the 7-bit right-justified slave address and does not include the R/W bit.

Return Value: None

Side Effects: None

void SMBusSlave_SetAlertResponseAddress(uint8 address)

Description: This function sets the Alert Response Address.

Parameters: uint8 address: Alert Response Address. This value can be any address between 0 and 127 (0x00 to 0x7F). The address is the 7-bit right-justified slave address and does not include the R/W bit. By default the address is set to 0xC0 to correspond with the SMBus specification.

Return Value: None

Side Effects: None

void SMBusSlave_SetSmbAlert(uint8 assert)

Description: This function sets the value to the SMBALERT# pin. As long as SMBALERT# is asserted, the component will respond to master read's to the Alert Response Address. The response will be the device's primary slave address. Depending on the mode setting, the component will automatically de-assert SMBALERT#, call the SMBusSlave_HandleSmbAlertResponse() API, or do nothing.

Parameters: uint8: value to set; options 0 or 1;

Return Value: None

Side Effects: None

void SMBusSlave_SetSmbAlertMode(uint8 alertMode)

Description: This function determines how the component responds to an SMBus master read at the Alert Response Address. When SMBALERT# is asserted, the SMBus master may broadcast a read to the global Alert Response Address to determine which SMBus device on the shared bus has asserted SMBALERT#.

In Auto mode, SMBALERT# is automatically de-asserted once the component acknowledges the Alert Response Address.

In Manual mode, the component will call the API SMBusSlave_HandleSmbAlertResponse() where user code (in a merge section) is responsible for de-asserting SMBALERT#.

In DO_NOTHING mode, the component will take no action.

Parameters: uint8: alertMode a byte that defines SMBALERT pin mode.

Value	Description
SMBusSlave_DO_NOTHING	Do nothing with SMBALERT# pin
SMBusSlave_AUTO_MODE	Automatically deassert SMBALERT# pin
SMBusSlave_MANUAL_MODE	User code is responsible for deasserting SMBALERT# pin

Return Value: None

Side Effects: None

void SMBusSlave_HandleSmbAlertResponse(void)

Description:	This function is called by the component when it responds to the Alert Response Address and the SMBALERT Mode is set to MANUAL_MODE. This function contains a merge code section where the user inserts code to run after the component has responded. For example, the user might update a status register and de-assert the SMBALERT# pin.
Parameters:	None
Return Value:	None
Side Effects:	None

TRANSACTION_STRUCT* SMBusSlave_GetNextTransaction(void)

Description:	This function returns a pointer to the next transaction record in the transaction queue. If the queue is empty, the function returns NULL. Only Manual Reads and Writes will be returned by this function, as the component will handle any Auto transactions on the queue. In the case of Writes, it is the responsibility of the user firmware servicing the Transaction Queue to copy the "payload" to the register store. In the case of Reads, it is the responsibility of user firmware to update the contents of the variable for this command in the register store. For both, call SMBusSlave_CompleteTransaction() to free the transaction record. Note that for Read transactions, the length and payload fields are not used for most transaction types. The exception to this is Process Call and Block Process Call, where the block of data from the write phase will be stored in the payload field.
Parameters:	None
Return Value:	Pointer the next transaction record
Side Effects:	None

uint8 SMBusSlave_GetTransactionCount(void)

Description:	This function returns the number of transaction records in the transaction queue.
Parameters:	None
Return Value:	uint8: Number of records in the transaction queue. The count will only be 0 or 1.
Side Effects:	None

void SMBusSlave_CompleteTransaction(void)

Description: This function causes the component to complete the currently pending transaction at the head of the queue. The user firmware transaction handler calls this function after processing a transaction. This alerts the component code to copy the register variable associated with the pending read transaction from the register store to the data transfer buffer so that the transfer may complete. It also advances the queue. Must be called for reads and writes.

Parameters: None

Return Value: None

Side Effects: None

uint8 SMBusSlave_GetReceiveByteResponse(void)

Description: This function is called by the component ISR to determine the response byte when it detects a "Receive Byte" protocol request. This function includes a merge code section where the user may insert their code to override the default return value of this function – which is 0xFF. This function will be called in ISR context. Therefore, user merge code must be fast, non-blocking, and may only call re-entrant functions.

Parameters: None

Return Value: uint8: User-specified status byte

Value	Description
SMBusSlave_RET_UNDEFINED	Default return status

Side Effects: None

void SMBusSlave_HandleBusError(uint8 errorCode)

Description: This function is called by the component whenever a bus protocol error occurs. Examples of bus errors would be: invalid command, data underflow, and clock stretch violation. This function is only responsible for the aftermath of an error since the component will already handle errors in a deterministic manner. This function is primarily for the purpose of notifying user firmware that an error has occurred. For example, in a PMBus device this would give user firmware an opportunity to set the appropriate error bit in the STATUS_CML register.

Parameters: uint8 errorCode:

Value	Description
SMBusSlave_ERR_READ_FLAG	Read Flag was incorrectly set
SMBusSlave_ERR_RD_TOO_MANY_BYTES	Host attempts to read too many bytes
SMBusSlave_ERR_WR_TOO_MANY_BYTES	Host attempts to write too many bytes
SMBusSlave_ERR_UNSUPPORTED_CMD	Received command is unsupported
SMBusSlave_ERR_INVALID_DATA	Received data is invalid
SMBusSlave_ERR_TIMEOUT	Bus reset timeout occured
SMBusSlave_ERR_CORRUPTED_DATA	Received PEC does not match

Return Value: None

Side Effects: None

uint8 SMBusSlave_StoreUserAll(const uint8 * flashRegs)

Description: This function saves the Operating register store to the User register store in Flash. The CRC field in the register store data structure is recalculated and updated prior to the save. This function does not perform storing anything to Flash by default. Instead it contains a merge region where the user can implement an algorithm for storing Operating register store to Flash.

Parameters: flashRegs: A pointer to a location in Flash where Operating register store (RAM) should be stored.

Return Value: uint8: status

Value	Description
CYRET_SUCCESS	Action completed successfully
CYRET_BAD_PARAM	Invalid parameter value

Side Effects: None

uint8 SMBusSlave_RestoreUserAll(const uint8 * flashRegs)

Description: This function verifies the CRC field of the User register store and then copies the contents of this register store to the Operating register store in RAM.

Parameters: flashRegs: A pointer to the User register store location in Flash.

Return Value: uint8: status

Value	Description
CYRET_SUCCESS	CRC matches and Operating register store was updated from the User register store (Flash)
CYRET_BAD_PARAM	Invalid parameter value
CYRET_BAD_DATA	Data is bad. CRC doesn't match

Side Effects: None

uint8 SMBusSlave_EraseUserAll(void)

Description: This function erases the User register store in Flash. The API does not erase the Flash by default. Instead it contains a merge region where the user can implement an algorithm to erase the contents of the User register store in Flash.

Parameters: None

Return Value: uint8: status

Value	Description
CYRET_SUCCESS	Action completed successfully

Or other user-determined non-SUCCESS status

Side Effects: None

uint8 SMBusSlave_RestoreDefaultAll(void)

Description: This function verifies the signature field of the Default register store and then copies the contents of the Default register store to the Operating register store in RAM.

Parameters: None

Return Value: uint8: One of following standard return statuses.

Value	Description
CYRET_SUCCESS	Action completed successfully
CYRET_BAD_DATA	Data is bad. CRC does not match

Side Effects: None

uint8 SMBusSlave_StoreComponentAll(void)

Description: This function updates the parameters of other components in the system with the current PMBus settings. Because this action is very application specific, this function consists almost entirely of a merge section. The only component provided firmware is a return value variable (retval) which is initialized to CYRET_SUCCESS and returned at the end of the function. The rest of the function must be user provided.

Parameters: None

Return Value: uint8: One of following standard return statuses.

Value	Description
CYRET_SUCCESS	Action completed successfully

Or other user-determined non-SUCCESS status.

Side Effects: None

uint8 SMBusSlave_RestoreComponentAll(void)

Description: This function updates the PMBus Operating register store with the current configuration parameters of other components in the system. Because this action is very application specific, this function consists almost entirely of a merge section. The only component provided firmware is a return value variable (retval) which is initialized to CYRET_SUCCESS and returned at the end of the function. The rest of the function must be user provided.

Parameters: None

Return Value: uint8: One of following standard return statuses.

Value	Description
CYRET_SUCCESS	Action completed successfully

Or other user-determined non-SUCCESS status.

Side Effects: None

float SMBusSlave_Lin11ToFloat (uint16 linear11)

Description: This function converts the argument "linear11" to floating point and returns it.

Parameters: uint16 linear11: A number in LINEAR11 format.

Return Value: float: The linear11 parameter converted to floating point

Side Effects: None

uint16 SMBusSlave_FloatToLin11 (float floatvar)

Description:	This function takes the argument "floatvar" (a floating point number) and converts it to a 16-bit LINEAR11 value (11-bit mantissa + 5-bit exponent), which it returns.
Parameters:	float floatvar: A floating point number
Return Value:	uint16: floatvar converted to LINEAR11
Side Effects:	None

float SMBusSlave_Lin16ToFloat(uint16 linear16, int8 inExponent)

Description:	This function converts the argument "linear16" to floating point and returns it. The argument Linear16 contains the mantissa. The argument inExponent is the 5-bit 2's complement exponent to use in the conversion.
Parameters:	uint16 linear16: The 16-bit mantissa of a LINEAR16 number.
	int8inExponent: The 5-bit exponent of a LINEAR16 number. Packed in the lower 5 bits. 2's Complement.
Return Value:	float: The parameters converted to floating point
Side Effects:	None

uint16 SMBus_FloatToLin16(float floatvar, int8 outExponent)

Description:	This function takes the argument "floatvar" (a floating point number) and converts it to a 16-bit LINEAR16 value (16-bit mantissa), which it returns. The argument outExponent is the 5-bit 2's complement exponent to use in the conversion.
Parameters:	floatfloatvar: A floating point number to be converted to LINEAR16.
	int8outExponent: User provided 5-bit exponent to use in the conversion.
Return Value:	uint16: The parameters converted to LINEAR16.
Side Effects:	None

Bootloader Support

The SMBus and PMBus Slave component can be used as a communication component for the Bootloader. For more information about the Bootloader, refer to the Bootloader component datasheet.

Note In order to communicate with a PSoC device using the SMBus communication interface for the Bootloader, an SMBus Bootloader Host is required. The Bootloader Host application provided with PSoC Creator does not support SMBus protocol.

The component provides a set of API functions for Bootloader use.

Function	Description
SMBusSlave_CyBtldrCommStart	Starts the SMBus and PMBus Slave component and enables its interrupt.
SMBusSlave_CyBtldrCommStop	Disables the SMBus and PMBus Slave component and disables its interrupt.
SMBusSlave_CyBtldrCommReset	Sets read and write I^{2}C buffers to the initial state and resets the slave status.
SMBusSlave_CyBtldrCommWrite	Allows the caller to write data to the bootloader host. This function manages polling to allow a block of data to be completely sent to the host device.
SMBusSlave_CyBtldrCommRead	Allows the caller to read data from the bootloader host. This function manages polling to allow a block of data to be completely received from the host device.

void SMBusSlave_CyBtldrCommStart(void)

Description: Starts the communication component and enables the interrupt. The read buffer initial state is full and the read always is 0xFFu. The write buffer is clear and ready to receive a command.

Parameters: None

Return Value: None

Side Effects: This function enables the component interrupt. If I^{2}C is enabled without the interrupt enabled, it could lock up the bus.

void SMBusSlave_CyBtldrCommStop(void)

Description: Disables the communication component and disables the interrupt.

Parameters: None

Return Value: None

Side Effects: None

void SMBusSlave_CyBtldrCommReset(void)

Description:	Sets buffers to the initial state and reset the statuses. The read buffer initial state is full and the read always is 0xFFu. The write buffer is clear and ready to receive a command.
Parameters:	None
Return Value:	None
Side Effects:	None

cystatus SMBusSlave_CyBtldrCommRead(uint8 * pData, uint16 size, uint16 * count, uint8 timeOut)

Description:	Requests that the provided size number of bytes are read from the host device and stored in the provided data buffer. Once the write is done count is updated with the number of bytes written. The timeOut parameter is used to provide an upper bound on the time that the function is allowed to operate. The host issues BOOTLOAD_WRITE command to transfer the block of data to the PSoC device.
Parameters:	uint8 *pData: Pointer to storage for the block of data to be read from the bootloader host uint16 size: Number of bytes to be read uint16 *count: Pointer to the variable to write the number of bytes actually read uint8 timeOut: Number of units in tens of milliseconds to wait before returning because of a timeout
Return Value:	cystatus: Returns CYRET_SUCCESS if no problem was encountered or returns the value that best describes the problem. For more information, see the "Return Codes" section of the *System Reference Guide*.
Side Effects:	None

cystatus SMBusSlave_CyBtldrCommWrite(const uint8 * pData, uint16 size, uint16 * count, uint8 timeOut)

Description:	Requests that the provided size number of bytes is written from the provided data buffer to the host device. Once the write is done count is updated with the number of bytes written. The timeOut parameter is used to provide an upper bound on the time that the function is allowed to operate. The host issues BOOTLOAD_READ command to read the block of data from the PSoC device.
Parameters:	const uint8 *pData: Pointer to the block of data to be written to the bootloader host uint16 size: Number of bytes to be written uint16 *count: Pointer to the variable to write the number of bytes actually written uint8 timeOut: Number of units in tens of milliseconds to wait before returning because of a timeout
Return Value:	cystatus: Returns CYRET_SUCCESS if no problem was encountered or returns the value that best describes the problem. For more information see the "Return Codes" section of the *System Reference Guide*.
Side Effects:	Temporarily disables the component interrupt when writing a block of data from the bootloader to the component transfer buffer. The communication is not running during this period of time

Macros

Macro	Description
SMBusSlave_FL_ADDR_TO_ROW(addr)	Extracts Flash row number from specified address
SMBusSlave_FL_ADDR_TO_ARRAYID(addr)	Extracts Flash array ID from specified address
SMBusSlave_SIZE_TO_ROW(size)	Calculates and returns the number of Flash rows required to store the number of data defined by ***size***
SMBusSlave_MAX_PAGES	Specifies the maximum number of pages for paged commands
SMBusSlave_NUM_COMMANDS	Total number of supported commands

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator components
- specific deviations – deviations that are applicable only for this component

This section provides information on component-specific deviations. Project deviations are described in the MISRA Compliance section of the System Reference Guide along with information on the MISRA compliance verification environment.

The SMBus Slave component has the following specific deviations:

MISRA-C:2004 Rule	Rule Class (Required/ Advisory)	Rule Description	Description of Deviation(s)
11.4	A	A cast should not be performed between a pointer to object type and a different pointer to object type.	Violated for the following reasons: To access a floating point variable as raw 32-bit value in FloatToLin11() and FloatToLin16() unit conversion APIs For byte access to/from 16-bit command fields of the register store data structure. For byte access to the register store structure when the CRC field of the register store is computed.
11.5	R	A cast shall not be performed that removes any const or volatile qualification from the type addressed by a pointer.	A cast is performed when a pointer to const is passed as the source buffer to memcpy() c51 library routine which expects void * type for this parameter.
17.4	R	Array indexing shall be the only allowed form of pointer arithmetic.	The component applies array subscripting to an object of pointer type to access command fields of the register store structure.
19.7	A	A function should be used in preference to a function-like macro.	Deviated for more efficient code.

This component has the following embedded components: Clock, I^{2}C, Pin, TCPWM. Refer to the corresponding component datasheets for information on their MISRA compliance and specific deviations.

Sample Firmware Source Code

PSoC Creator provides numerous example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or File menu. As needed, use the Filter Options in the dialog to narrow the list of projects available to select.

Refer to the "Find Example Project" topic in the PSoC Creator Help for more information.

Functional Description

The theory of operation of this component is very similar to an I^2C Slave component. All references the SMBus specification refer to the SMBus specification version 2.0. Key functionalities of the component are highlighted in this section.

I^2C Physical Layer

The physical layer of the SMBus/PMBus slave component is based on the I^2C protocol. The three main differences that impact this component are:

The SMBus specification mandates that the component must reset and release the SCL and SDA lines if the SCL signal is detected stuck low for 25 ms. This is described in more detail in the DC and AC Electrical Characteristics section later in this document.

The SMBus Specification mandates that the component must not stretch the clock more than 25 ms (cumulative) in any given transfer. The slave is allowed to delay transfers when it is busy by pulling SCL low (clock stretching) provided that the cumulative stretch time in any transaction does not exceed 25 ms.

Addition of an SMBALERT# pin to notify the host that the device needs attention.

SMBus/PMBus Addressing

Every SMBus/PMBus slave device has an I^2C address. The following addresses are reserved for specific SMBus usage and must not be used as the generic slave address for an SMBus/PMBus slave.

Slave Address (Bits 7:1)	R/W# bit (Bit 0)	Comment
0000 000	0	General Call Address
0000 000	1	START byte
0000 001	X	CBUS address
0000 010	X	Reserved for different bus format
0000 011	X	Reserved for future use
0000 1XX	X	Reserved for future use
0101 000	X	Reserved for ACCESS bus host
0110 111	X	Reserved for ACCESS bus default address
1111 0XX	X	Reserved for 10-bit slave addressing
1111 1XX	X	Reserved for future use
0001 000	X	Reserved for SMBus Host
0001 100	X	SMBus Alert Response Address
1100 001	X	SMBus Device Default Address

General Call Address

The general call address is for addressing every device connected to the bus. The component responds to the General Call Address (00h) as well as its own physical address.

Note On PSoC 3 and PSoC 5LP, the Address decode parameter must be set to Software for the component response to the General Call Address.

SMBus Alert Response Address

The component responds to its primary address and the SMBus Alert Response Address if the SMBALERT# pin option is enabled.

SMBus Device Default Address

The SMBus Device Default Address is reserved for use by the SMBus Address Resolution Protocol, which is not supported by the component.

SMBus/PMBus Protocols

Nine different protocols are defined in the SMBus Specification. A summary of these protocols and their support statuses are shown below. Refer to section 5.5 “Bus Protocols” of the SMBus specification for a detailed description of each protocol format.

Protocol	Support Status
Quick Command	Not Supported
Send Byte	Supported
Receive Byte	Supported (SMBus mode)
Write Byte/Word	Supported
Read Byte/Word	Supported
Process Call	Supported
Block Write/Read	Supported
Block Write/Block Read Process Call	Supported
SMBus Host Notify Protocol	Not Supported

Key concepts that characterize the SMBus/PMBus component are as follows.

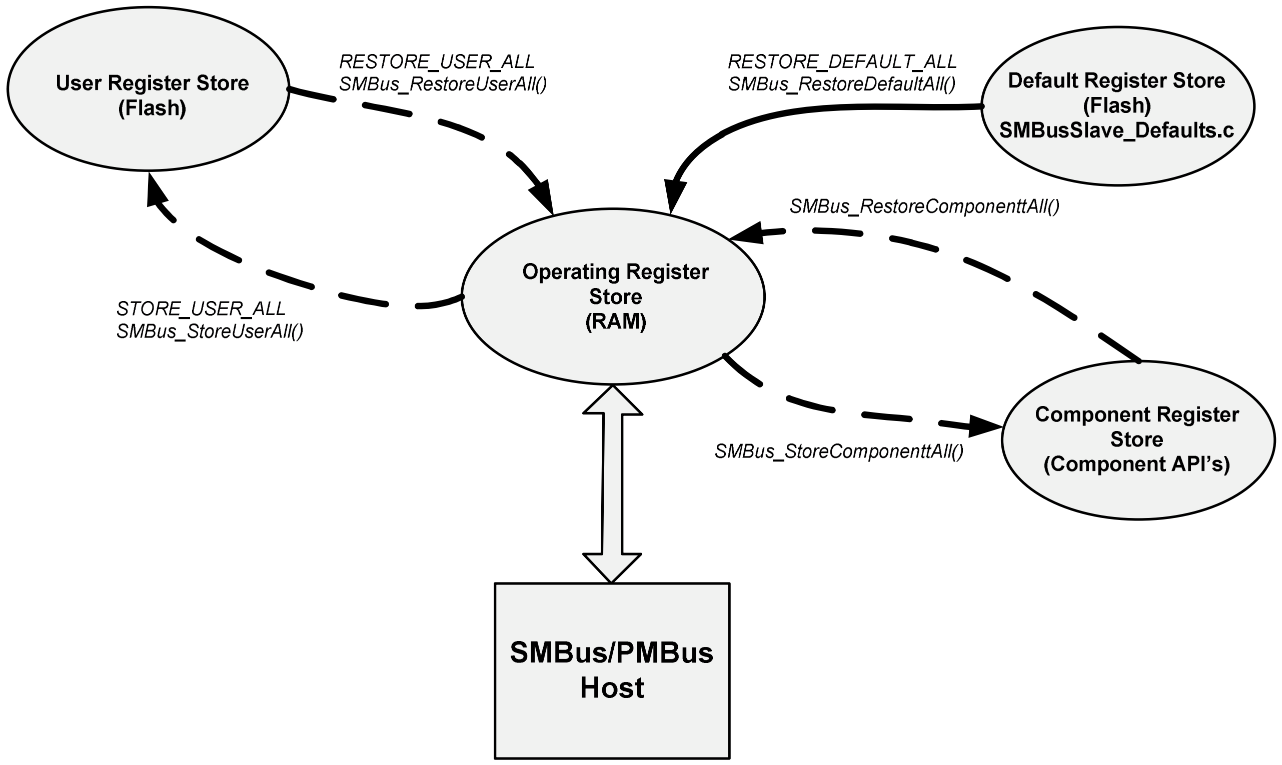
Neither the SMBus nor PMBus use the concept of a sub-address or an I^{2}C register map. All transfers are command based. Immediately following the slave device address is a command code. Commands can be write-only, read-only or read/write. The SMBus specification does not define any of the command codes and their read/write restrictions; they are up to the user to define. However, the PMBus Specification Part II, Appendix I does define all 256 possible command codes; 46 of these are up to the user to define (called manufacturer-specific commands).

SMBus and PMBus are little-endian protocols (i.e., the least significant byte of a multi-byte data-type is transmitted first on the bus).

Register Stores Concept

According to the PMBus Specification II 5.4.2 – "Every Parameter That Can Be Written Must Be Readable". In general, any command that accepts a value for writing must also return that value when read. For this purpose, a concept of Register Stores is used.

Figure 1. Register Stores Concept


User Register Store (Flash)
RESTORE_USER_ALL
SMBus_RestoreUserAll()
RESTORE_DEFAULT_ALL
SMBus_RestoreDefaultAll()
Default Register Store (Flash) SMBusSlave_Defaults.c
SMBus_RestoreComponenttAll()
STORE_USER_ALL
SMBus_StoreUserAll()
Operating Register Store (RAM)
SMBus_StoreComponenttAll()
Component Register Store (Component API's)
SMBus/PMBus Host

Operating Register Store

This is the RAM version of the register store. Runtime SMBus/PMBus commands modify this version of the register store. Since this register store is located in RAM, its contents are assumed to be invalid at reset.

Default Register Store

This is the Flash version of the register store, containing default values for all of the SMBus/PMBus parameters. The parameter values in the Default register store are fixed at compile/link time, but the user is responsible for providing the default values. Open the SMBusSlave_Defaults.c file, copy the parameters from the comment block and paste them into the merge region below. This will be store the default values in SMBusSlave_regsDefault and the Operating register store will be initialized with these parameters at every startup. The Operating register store can be initialized with default values during component operation using SMBus_RestoreDefaultAll() API. This function can be used in user handler for RESTORE_DEFAULT_ALL (0x12) PMBus command.

User Register Store

This is another Flash version of the register store. Unlike the Default register store, which is essentially read-only, the User register store can be updated based on the current contents of the Operating register store. Two API functions, SMBusSlave_StoreUserAll() and SMBusSlave_RestoreUserAll(), are provided to work with this register store. These functions perform CRC calculations that are used to check data validity at store/restore.

As storing data into Flash is very implementation specific, the default component does not provide the code to storing anything into Flash. SMBusSlave_StoreUserAll() contains a user-editable code section to implement a method of storing data into Flash.

To design your own method of storing data into Flash, refer to the *System Reference Guide*. Chapter 9 has basic information, various functions, and a set of macros for working with Flash.

Component Register Store

The primary motivation for the component register store is to allow PMBus to extract parameters from and configure standard PSoC Creator components (thus, the name – Component register store). Creator components have their own configuration parameters, which are usually set up with the component customizer. These parameters are accessed at runtime via component specific SMBusSlave_StoreComponentAll()/SMBusSlave_RestoreComponentAll() APIs. The component parameters accessible via the API's comprise the component register store. At startup, user PMBus firmware may want to:

- Update the other component settings based on PMBus parameters stored in User or Default register stores, or
- Update the PMBus Operating register store based on the component settings.

Transaction Queue

Any SMBus/PMBus commands not designated as AUTO must be handled in your main program context in a timely manner. To accomplish this, the component will maintain a transaction queue so your code may process bus transactions out of buried component ISR context.

Wherever the command of type "Manual" is detected, it is recorded in the Transaction Queue and should be handled by user code. The Transaction Queue record has the following structure:

```
typedef struct
{
    uint8 read;         /* r/w flag - 1=read 0=write */
    uint8 commandCode; /* SMBus/PMBus command code */
    uint8 page;         /* SMBus/PMBus page */
    uint8 length;       /* bytes transferred */
    uint8 payload[65]; /* payload for the transaction */
} TRANSACTION_STRUCT
```

The following are descriptions for the fields:

- “read” is a flag that indicates the type of command received either Read or Write
- “commandCode” is a 1-byte command code of the currently received command
- “page” is a page number for the currently received command

  It is only applicable for Paged commands. For Common commands this field is zero.
- "length"
  - For the Write command, “length” specifies data length in bytes for the currently received command.
  - For the Read command, “length” specifies the number of bytes to be sent.
  - For block commands, "length" includes the "byte count" byte.
- "payload"
  - For Write commands, “payload” contains the received data for the current command. User code is responsible for updating the Operating register store and then calling SMBusSlave_CompleteTransaction().
  - For Read commands, "payload" isn't used. User code is responsible for updating the variable for this command in the Operating register store and then calling SMBusSlave_CompleteTransaction().

Each time a manual command is received, the component will stretch the clock until the pending transaction is handled and SMBusSlave_CompleteTransaction() is called. For the Read command, the clock stretching begins after repeated starts, waiting for the user code to provide the response for the external host. For the Write command, the clock stretching begins after the data for current command is received. When the clock stretching begins, the internal timer starts to count the 25 ms timeout. If the SMBusSlave_CompleteTransaction() is not called before the timeout occurrence, the component will reset the bus. The remaining record in the Transaction Queue will be invalidated.

PAGE command

The PMBus PAGE (Code 0x00) command (PMBus Part II – Section 11.10) allows the access of multiple logical PMBus devices at the same PMBus slave device address. For example, a PMBus power supervisor that controls multiple power rails could provide access to the commands/parameters for each rail on its own page. The page can be thought of as an index into an array of commands/registers. Once the page is set via the PAGE command, the page setting is persistent until set again by another PAGE command.

In PMBus mode, the component has built-in support for the PAGE command. In SMBus mode, the user has the option to enable the PAGE command and specify the command code to use for PAGE.

If the PAGE command is enabled, the valid range of the PAGE command values are between 1 and 64, and must be within the Max Page setting determined by the user. The exception is the "All Pages" wild card setting, which is 0xFF. The "All Pages" wild card setting is only valid for write transactions, and must always be handled in manual mode. If the PAGE is set to 0xFF, the following transactions are treated as errors:

- An attempt to Read from a Paged command
- An attempt to Write to a Paged command that is configured as Auto

Even in a PMBus device with multiple pages, some commands are not-dependent on the current page and are always handled in the same way regardless of the PAGE setting. For example, the PMBUS_REVISION command, which returns the PMBus version that the device supports, is common to all pages. Thus, there is a concept of Page commands and Common commands. For each SMBus/PMBus command, the customizer allows the user to specify whether the command is Paged or Common. When a command is marked as Paged, the component defines an array in the Register Store for that command.

Bootloader Commands

When the component is placed in a "Bootloader" project, two additional commands become available in the Configure dialog on the **Custom Commands** tab. These commands are BOOTLOAD_READ and BOOTLOAD_WRITE with default command codes of 0xFD and 0xFC, respectively. They add a capability to the component to act as a communication component for the Bootloader component.

These two commands interact with the bootloader component placed on a design schematic. After placing the bootloader component, select "Custom interface" in the bootloader component Configure dialog.

Note The bootloader commands use Block write/read bus protocol as defined in the SMBus specification. According to the specification, a Block Read or Write is allowed to transfer a maximum of 32 data bytes, when the component allows transferring a maximum of 64 data bytes in one bootloader command. This does not prevent the component operation with legacy SMBus host devices and adds more flexibility to a design. In general any packet length of up to 64 bytes is supported for the bootloader commands. 64 bytes does not include a byte count for data bytes transferred in a packet.

Data Transmission and Content Faults

The component responds to the transmission and content faults defined in the PMBus Specification in a deterministic manner and after that calls the HandleBusError() function to notify the user firmware. This would give user firmware an opportunity to set the appropriate error bit in the STATUS_CML register. The following bus errors are reported by the component:

Corrupted Data

Corrupted data can only be detected if Packet Error Checking (PEC) is enabled in the customizer (Disabled by default). Whenever the component detects that the received and calculated PEC bytes do not match, it will respond as follows:

- NACK the PEC byte
- Not respond to or act upon the received command
- Ignore the received command code and any received data
- Call HandleBusError(ERR_CORRUPTED_DATA) to notify the user firmware

Host Sends or Reads Too Few Bytes

For each supported command, the component expects a fixed number of bytes to be written. If the host transmits fewer bytes than expected, the component completely ignores the command and takes no action.

Host Sends Too Many Bytes

For each supported command, the component expects a fixed number of bytes to be written. If the host sends more bytes than expected, this is a data transmission fault.

Sending a PEC byte to the component if the Packet error checking is disabled is included in this fault.

When the component detects this fault, it will respond as follows:

- NACK all of the unexpected bytes as they are received (until the next STOP condition is received)
- Ignore the received command code and any received data
- Call HandleBusError(ERR_WR_TOO_MANY_BYTES) to notify the user firmware

Reading Too Many Bytes

For each supported command, the component expects a fixed number of bytes to be read. If while reading from the component, the host tries to read more bytes than the component is expecting to send, this is a data transmission fault.

Trying to read a PEC byte if the Packet error checking is disabled is included in this fault.

When the component detects this fault, it will respond as follows:

- Send all ones (FFh) as long as the host keeps clocking and acknowledging
- Call HandleBusError(ERR_RD_TOO_MANY_BYTES) to notify the user firmware

Improperly Set Read Bit in the Address Byte

This error occurs when the Read bit is set on the first Address after the START. This is illegal for PMBus since all transactions begin with a Write. However, if the component is in SMBus mode, this is a valid “Receive Byte” protocol transaction. When the component detects this fault, it will respond as follows:

- ACK the address byte as all SMBus devices must ACK their own address
- Send all ones (FFh) as long as the host keeps clocking and acknowledging
- Call HandleBusError(ERR_READ_FLAG) to notify the user firmware

Unsupported Command Code

If the component receives a command that it does not support, it will respond as follows:

- NACK the unsupported command code and all data bytes received before the next STOP condition
- Ignore the received command code and any received data
- Call HandleBusError(ERR_UNSUPPORTED_CMD) to notify the user firmware

Host Reads from a Write Only Command

When the host tries to read from a write only command, this is a data content fault. The component will respond as follows:

- Send all ones (FFh) as long as the host keeps clocking and acknowledging
- Call HandleBusError(ERR_INVALID_DATA) to notify the user firmware

Host Writes to a Read Only Command

When the host tries to write to a read only command, this is a data content fault. The component will respond as follows:

- Ignore the received command code and any received data
- Call HandleBusError(ERR_INVALID_DATA) to notify the user firmware

Host Reads from a Paged Command if the PAGE is 0xFF

An attempt to read from a paged command if the page is set to “All pages” wildcard (0xFF) is a data content fault. The component will respond as follows:

- Send all ones (FFh) as long as the host keeps clocking and acknowledging

- Call HandleBusError(ERR_INVALID_DATA) to notify the user firmware

Host Writes to a Paged Command if the PAGE is 0xFF

An attempt to write to a paged command if the page is set to “All pages” wildcard (0xFF) is considered as a data content fault if the command is configured for Auto Write mode. The component will respond as follows:

- Ignore the received command code and any received data
- Call HandleBusError(ERR_INVALID_DATA) to notify the user firmware

SMBus Timeout

When the component detects any single clock held low longer than the timeout period (tTIMEOUT = 25 ms), the component resets its communication and releases the bus. The component notifies the user firmware by calling HandleBusError(ERR_INVALID_DATA) API prior re-enabling its communication interface.

User Provided Code Sections

Several component functions should take very application specific actions, for example:

- storing/restoring the content of the Operating register store to/from the User register store in Flash
- providing the initial values for the Default register store parameters
- setting the appropriate error bit in the STATUS_CML register as response to data transmission and content faults

Therefore these functions must be user provided. Additional actions may also be required during a data transmission in the component ISR.

To accomplish this goal, the API files that are automatically generated by PSoC Creator provide user-editable code sections with embedded comments on the action needed. All of these user-editable code sections can be found in the *SMBusSlave.c, SMBusSlave_Defaults.c*, and *SMBusSlave_INT.c* API source code files, located in the **Generated_Source/<Device>** [2]**/ SMBusSlave** folder in the **Workspace Explorer** window. **Note** that the file name and folder name examples assume that the instance name of the component is set to **SMBusSlave**.

The following are summaries of files with the user-editable code sections.

2. These are shown as “PSoC3”, “PSoC4”, or “PSoC5” depending on the device family targeted.

SMBusSlave.c

There are user-editable code sections at the top of the file to include header files from components whose APIs are called from within this file and define global variables respectively. Other sections are local to the API functions listed in the table below.

Function	User code purpose
SMBusSlave_HandleSmbAlertResponse()	The aftermath of a component response to the Alert Response Address. For example, update of the STATUS_CML register and de-assert the SMBALERT# pin.
SMBusSlave_GetReceiveByteResponse()	Providing a data byte to respond to a "Receive Byte" protocol request.
SMBusSlave_HandleBusError()	The aftermath of a data transmission and content fault, e.g. setting an error bit in the STATUS_CML register for a PMBus device.
SMBusSlave_StoreUserAll()	Storing the Operating register store to the User register store in Flash.
SMBusSlave_EraseUserAll()	Erasing the User register store in Flash.
SMBusSlave_StoreComponentAll()	Updating the parameters of other components in the system according to the content of the Operating register store.
SMBusSlave_RestoreComponentAll()	Updating the content of the Operating register store according to the parameters of other components in the system.

To design your own method of storing data into Flash, refer to the *System Reference Guide*. Chapter 9 has basic information and descriptions of functions that work with Flash.

SMBusSlave_Defaults.c

There is a user-editable code section for providing the initial parameter values in the Default register store that are fixed at compile/link time. Refer to Default Register Store section for details.

SMBusSlave_INT.c

As with the SMBusSlave.c file, there is a user-editable section at the top of the file for including header files and variable definitions. Also, there are two more sections in the interrupt service routine. One section is just after interrupt entry and another where the received address and the device address match.

References

- System Management Bus (SMBus) Specification (Version 2.0)
- PMBus™ Power System Management Protocol Specification Part I – General Requirements, Transport and Electrical Interface (Revision 1.2)
- PMBus™ Power System Management Protocol Specification Part II – Command Language (Revision 1.2)

Resources

On PSoC 3 and PSoC 5LP, the SMBus/PMBus component resource usage is mostly dependent on the implementation type of the I^2C interface. That is, the I^2C can either be implemented in the UDB array or use the dedicated I^2C Fixed Function (FF) block. On PSoC 3 and PSoC 5LP the component utilizes the following resources.

Configuration	Resource Type					
	Datapath Cells	**Macrocells**	**Status Cells**	**Control Cells**	**I^2C Fixed Blocks**	**Interrupts**
SMBus / PMBus (UDB)	2	29	3	4	-	2
SMBus / PMBus (FF)	1	5	1	2	1	2

On PSoC 4 there are no options for the implementation type of the I^2C interface. The component utilizes a Serial Communication Block (SCB), a Timer/Counter/PWM Block (TCPWM) and two Interrupts.

API Memory Usage

The component memory usage varies significantly depending on the compiler, device, number of APIs used and component configuration.

Furthermore, the memory usage is dependent on the number of commands, the number of pages, and the features enabled. Additional commands consume SRAM for the runtime register store and Flash for the default register store and command lookup table. That is, the memory usage is expressed as the size of base component code and the memory usage for additional commands.

The following table provides the memory usage for all APIs available in the given component configuration. The usage for each individual feature is presented as the additional memory space required when the feature is enabled.

The measurements have been done with an associated compiler configured in Release mode with optimization set for Size. For a specific design, the map file generated by the compiler can be analyzed to determine the memory usage.

Configuration	PSoC 3 (Keil_PK51)		PSoC 4 (GCC)		PSoC 5LP (GCC)	
	Flash Bytes	**SRAM Bytes**	**Flash Bytes**	**SRAM Bytes**	**Flash Bytes**	**SRAM Bytes**
SMBus/PMBus core	3340 + CMD_{FLS}	185 + CMD_{RAM}	2595 + CMD_{FLS}	195 + CMD_{RAM}	2400 + CMD_{FLS}	185 + CMD_{RAM}
SMBALERT# support	+102	+2	+140	+2	+110	+2
PEC support	+391	+1	+320	+1	+316	+1

Configuration	PSoC 3 (Keil_PK51)		PSoC 4 (GCC)		PSoC 5LP (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Bootloader support	+376	+131	+326	+131	+318	+131

$$CMD_{RAM} = REGS\_SIZE = \sum_{i=1}^{N} cmd\_size_i \cdot page\_num_i$$

where:

- REGS_SIZE is the size of the Register Store
- N is the number of commands
- cmd_size is the size of command in bytes
- page_num is the number of pages for this command. For non-paged commands page_num is 1.

Command size is dependent on the command type as follows:

Command type	Command size
Send Byte	0
Read/Write Byte	1
Read/Write Word, Process Call	2
Block Write/Read, Block Process Call	size from customizer + 1

$$CDM_{FLS} = REGS\_SIZE + CMD\_LUT$$

where:

- REGS_SIZE is the size of Register Store and is calculated as for SRAM usage
- CMD_LUT is command lookup table size and is calculated as follows:
  - CMD_LUT = LUT_ENTRY * N

    where:

    LUT_ENTRY = 6 bytes for Keil_PK51 and 7 bytes for GCC 4.7.3

DC and AC Electrical Characteristics

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted. Specifications are valid for 1.71 V to 5.5 V, except where noted.

DC Characteristics

Parameter	Description		Min	Typ	Max	Unit
V_{DD}	Nominal bus voltage [3]		1.71	–	5.5	V
V_{IH}	Input high voltage (CMOS Input)		0.7 x V_{DD}	–	–	V
V_{IL}	Input low voltage (CMOS Input)		–	–	0.3 x V_{DD}	V
V_{IH}	Input high voltage (LVTTL Input, V_{DD} < 2.7 V)		0.7 x V_{DD}	–	–	V
V_{IH}	Input high voltage (LVTTL Input, V_{DD} ≥ 2.7 V)		2.0	–	–	V
V_{IL}	Input low voltage (LVTTL Input, V_{DD} < 2.7 V)		–	–	0.3 x V_{DD}	V
V_{IL}	Input low voltage (LVTTL Input, V_{DD} ≥ 2.7 V)		–	–	0.8	V
V_{OL}	Output low voltage	(I_{OL} = 8 mA at 3 V V_{DD})	–	–	0.6	V
		(I_{OL} = 3 mA at 3 V V_{DD})	–	–	0.4	V
$I_{LEAK-PIN}$	Input Leakage per device pin		–	–	2	nA
I_{PULLUP}	Current sinking	(V_{OL} = 0.6V at 3 V V_{DD})	8	–	–	mA
		(V_{OL} = 0.4V at 3 V V_{DD})	3	–	–	mA
C_i	Input Capacitance		–	–	7	pF
I_{DD} [4]	Component current consumption					
	PSoC 3/5LP	FF I^2C at 100 kHz	–	620	–	µA
		FF I^2C at 400 kHz	–	860	–	µA
		UDB I^2C at 100 kHz	–	440	–	µA
		UDB I^2C at 400 kHz	–	720	–	µA
	PSoC 4	SCB I^2C at 100 kHz	–	30	–	µA
		SCB I^2C at 400 kHz	–	170	–	µA

3. V_{DD} is listed as V_{DDIO} for PSoC 3 / PSoC 5LP and V_{DDD} for PSoC 4.

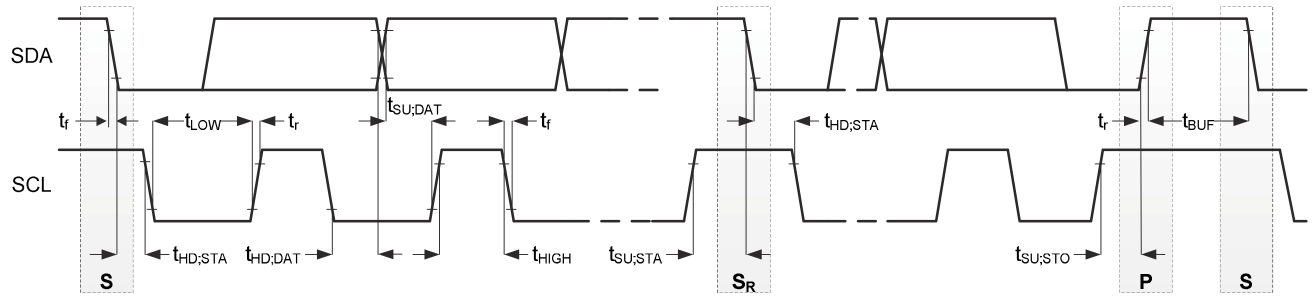
4. Device IO and clock distribution current are not included. The values are at 25 °C. Data was measured at BUS_CLK set to 24 MHz

AC Characteristics

Parameter	Description	Min	Typ	Max	Unit
SMBus Mode					
f_{SMB}	SMBus operating frequency	10	–	100	kHz
t_{BUF}	Bus free time between a stop and start condition	4.7	–	–	µs
t_{HD_STA}	Hold time after a (Repeated) start condition	4.0	–	–	µs
t_{SU_STA}	Setup time for a repeated start condition	4.7	–	–	µs
t_{SU_STO}	Setup time for stop condition	4.0	–	–	µs
t_{HD_DAT}	Data hold time	300	–	–	ns
t_{SU_DAT}	Data setup time	250	–	–	ns
t_{LOW}	Low period of the SCL clock	4.7	–	–	µs
t_{HIGH}	High period of the SCL clock	4.0	–	–	µs
PMBus Mode					
f_{SMB}	SMBus operating frequency	100	–	400	kHz
t_{BUF}	Bus free time between a stop and start condition	1.3	–	–	µs
t_{HD_STA}	Hold time after a (Repeated) start condition	0.6	–	–	µs
t_{SU_STA}	Setup time for a repeated start condition	0.6	–	–	µs
t_{SU_STO}	Setup time for stop condition	0.6	–	–	µs
t_{HD_DAT}	Data hold time	300	–	–	ns
t_{SU_DAT}	Data setup time	100	–	–	ns
t_{LOW}	Low period of the SCL clock	1.3	–	–	µs
t_{HIGH}	High period of the SCL clock	0.6	–	–	µs
Common Parameters					
f_{CLOCK}	Component input clock frequency	–	$16 \times f_{SMB}$	–	kHz
$t_{TIMEOUT}$	Detect clock low timeout	25	–	35	ms
t_{LOW_SEXT}	Cumulative clock low extend time	–	–	25	ms
t_F	Clock/Data Fall Time (3.3V V_{DDIO} C_{LOAD} = 25 pF)	–	–	12	ns
t_R	Clock/Data Rise Time (3.3V V_{DDIO} C_{LOAD} = 25 pF)	–	–	12	ns

Parameter	Description	Min	Typ	Max	Unit
t_{POR} [5]	Time in which a device must be operational after power-on reset	–	–	200	µs
t_{RESET}	Reset pulse width	–	2	–	$1/f_{CLOCK}$

Figure 2. Data Transition Timing Diagram



Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
3.0	Added PSoC 4 support	New device support
	Added Packed error checking support	Improved reliability and communication robustness
	Code optimization and refactoring	Improved code efficiency and reduced memory footprint
	Corrected a defect with processing of unpaged commands when a page was set to all pages (0xFF).	Component ignored the received command code and any received data for unpaged commands when a page was set to all pages (0xFF).
	Corrected a defect with Process Call protocol commands	Component ignored the received data and incorrectly generated Reading Too Many Byte fault condition for this type of transactions. Problem solved.
	Improved the mechanism of responses to data transmission faults	Improved the efficiency of responses to data transmission faults

5. Based on device characterization (Not production tested). The measurement was done with the sole SMBusSlave component in a design and includes the time from VDDD/VDDA/VCCD/VCCA ≥ PRES to the component is in operational state (no PLL used, slow IMO boot mode at 12 MHz).

Version	Description of Changes	Reason for Changes / Impact
	Corrected a defect with incomplete block write commands that may cause data corruption in the register store.	For each supported command, the component expects a fixed number of bytes to be written. If the host transmits fewer bytes than expected, the component completely ignores the command and takes no action.
	Added detection of Reading Too Many Bytes fault for Receive Byte protocol.	Improved error detection and reporting.
	Added detection of Reading Too Many Bytes fault for Receive Byte protocol.	Improved error detection and reporting.
	Corrected the number of bytes expected for SMBALERT_MASK command to comply with PMBus specification.	The number of bytes for SMBALERT_MASK did not correspond to the command size specified in PMBus specification.
	Corrected a defect with de-assertion of SMBALERT# pin in Auto mode.	SMBALERT# pin did not automatically de-assert after the host queried the device at the Alert Response Address.
	Added support of the General Call address (00h).	The component may respond to the General Call address (00h) as well as its own physical address.
	Added Manufacturer value to the command Format parameter in the Configure dialog	Allows selection of the manufacturer-specific data format value for a command.
	Datasheet update and corrections.	To reflect all changes in version 3.0.
2.20	The issue that caused the component to transmit "junk" data (instead of FFs) while trying to read write-only command was fixed.	The issue was caused by incorrect condition of handling read part of a command inside component ISR.
	Support of PSoC 5 family devices was removed from the component.	
	API Memory Usage table was updated with new values.	
	The description of Transaction Queue was added.	Information about the Transaction Queue is required for handling of the Manual type commands.
	Fixed the issue related to erroneous setting the page number to maximum page number value.	Maximum page number should be maximum page number value minus one as the page indexing starts with zero.
	Erroneous presence of clock input was fixed.	When the Fixed Function implementation of I^2C was selected in the component customizer the component must not expose the clock input on the symbol as it can only use internal clock in this mode.
	The label "Actual data rate" on the General tab of the customizer was changed to "Attainable data rate".	
2.10	The issue related to inability of changing the PAGE to 0xFF ("all pages" wildcard) was fixed.	

Version	Description of Changes	Reason for Changes / Impact
2.0	The PMBus Register store was made to be always declared using arrays for paged commands even when the user selects only 1 page in the design.	This was an error that might lead to code restructuring.
	Added "Pages" column to the Custom Commands table.	This is a new parameter that defines the page number for specific custom command.
	Fixed issue with incomplete block writes transactions.	The code was changed to verify if the number of bytes specified by "Byte count" field equals to number of received data bytes. Previously code verified if number of data equals to the "Size" parameter for specific block command that is entered by user in the component customizer.
	Removed compilation error that is occurred in case when SMBALERT pin was left unconnected.	
	Fixed issue which caused erroneous generation of bus error in while processing of page indexed, manual command when page was set to "all pages" wildcard.	
	Restricted slave address from using addresses reserved for specific SMBus usage.	Component did not validate the address.
	Fixed minor issues.	
	The new function was added - SMBusSlave_EraseUserAll().	
1.10	Added MISRA Compliance section.	The component was not verified for MISRA compliance.
	Updated SMBus and PMBus Slave with the latest version of the I^2C and Control Register components.	
1.0	First release	