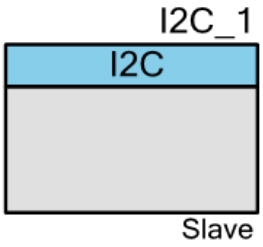
**Please note that Cypress is an Infineon Technologies Company.**
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of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

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

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

# I2C (SCB_I2C_PDL)
2.0

## Features




- Industry-Standard NXP I²C bus interface
- Supports slave, master [1] and master-slave operation
- Supports data rates of 100/400/1000 kbps
- Hardware Address Match, multiple addresses
- Wake from Deep Sleep on Address Match

## General Description

The SCB_I2C_PDL Component supports I²C slave, master, and master-slave operation configurations. The I²C bus is an industry-standard, two-wire hardware interface developed by Philips. The master initiates all communication on the I²C bus and supplies the clock for all slave devices.

The SCB_I2C_PDL Component supports standard clock speeds up to 1000 kbps. It is compatible with I²C Standard-mode, Fast-mode, and Fast-mode Plus devices as defined in the NXP I²C-bus specification. The SCB_I2C_PDL Component is compatible with other third-party slave and master devices.

The SCB_I2C_PDL Component is a graphical configuration entity built on top of the cy_scb driver available in the Peripheral Driver Library (PDL). It allows schematic-based connections and hardware configuration as defined by the Component Configure dialog.

### When to Use a SCB_I2C_PDL Component

The SCB_I2C_PDL Component is an ideal solution when networking multiple devices on a single board or small system. The system can be designed with a single master and multiple slaves, multiple masters, or a combination of masters and slaves.

---

1 Master modes provide all functionality necessary to work in a multi-master environment.

## Definitions

- $I^2C$ – The Inter Integrated Circuit ($I^2C$) bus is an industry-standard, two-wire hardware interface developed by Philips.
- SCB – Serial Communication Block. It supports $I^2C$, SPI, and UART interfaces.

## Quick Start

1. Drag an I2C (SCB) Component from the Component Catalog *Cypress/Communications/I2C* folder onto your schematic (placed instance takes the name I2C_1).
2. Double-click to open the Configure dialog.
3. Select **Data rate** at which the $I^2C$ interface is expected to communicate on the **Basic** tab. This is enough for master mode but for slave mode, the **Slave Address** must be configured.
4. Open the Design-Wide Resources Pin Editor, and assign the scl and sda pins for your design. Note that the choice of pins that can be used for the I2C interface is limited.
5. Build the project in order to verify the correctness of your design. This will add the required PDL modules to the Workspace Explorer, and generate configuration data for the I2C_1 instance.
6. In the *main.c* file, initialize the peripheral and start the application

   Slave:

```
/* Implement ISR for I2C_1 */
void I2C_1_Isr(void)
{
     Cy_SCB_I2C_Interrupt(I2C_1_HW, &I2C_1_context);
}

/* Allocate TX and RX buffers */
#define BUFFER_SIZE (128UL)
uint8_t rdBuffer[BUFFER_SIZE];
uint8_t wrBuffer[BUFFER_SIZE];

/* Initialize SCB for I2C operation with GUI selected settings */
(void) Cy_SCB_I2C_Init(I2C_1_HW, &I2C_1_config, &I2C_1_context);

/* Hook interrupt service routine and enable interrupt */
Cy_SysInt_Init(&I2C_1_SCB_IRQ_cfg, &I2C_1_Isr);
NVIC_EnableIRQ(I2C_1_SCB_IRQ_cfg.intrSrc);

/* Configure write and read buffer */
Cy_SCB_I2C_SlaveConfigReadBuf (I2C_1_HW, rdBuffer, BUFFER_SIZE, &I2C_1_context);
Cy_SCB_I2C_SlaveConfigWriteBuf(I2C_1_HW, wrBuffer, BUFFER_SIZE, &I2C_1_context);

/* Enable I2C */
Cy_SCB_I2C_Enable(I2C_1_HW);

/* Check slave status or use callback to get notification about
* communication with master. */
```

Master (uses I²C High-Level communication cy_scb driver functions; refer to the Interrupt Service Routine section):

```
/* Implement ISR for I2C_1 */
void I2C_1_Isr(void)
{
    Cy_SCB_I2C_Interrupt(I2C_1_HW, &I2C_1_context);
}

/* Allocate buffer */
#define BUFFER_SIZE (128UL)
uint8_t buffer[BUFFER_SIZE];

cy_stc_scb_i2c_master_xfer_config_t transaction;

/* Initialize SCB for I2C operation */
(void) Cy_SCB_I2C_Init(I2C_1_HW, &I2C_1_config, &I2C_1_context);

/* Configure desired data rate.
* Note that internally configured clock is utilized.
*/
(void) Cy_SCB_I2C_SetDataRate(I2C_1_HW, I2C_1_DATA_RATE_HZ, I2C_1_CLK_FREQ_HZ);

/* Hook interrupt service routine and enable interrupt */
Cy_SysInt_Init(&I2C_1_SCB_IRQ_cfg, &I2C_1_Isr);
NVIC_EnableIRQ(I2C_1_SCB_IRQ_cfg.intrSrc);

/* Enable I2C */
Cy_SCB_I2C_Enable(I2C_1_HW);

/* Configure transaction */
transaction.slaveAddress = 0x08u;
transaction.buffer       = buffer;
transaction.bufferSize   = BUFFER_SIZE;
transaction.xferPending  = false;

/* Master: request to execute transaction */
(void) Cy_SCB_I2C_MasterRead(I2C_1_HW, &transaction, &I2C_1_context);

/* Check master status or use callback to get notification about
* transaction completion.
*/
```

7. Build and program the device.

# Input/Output Connections

This section describes the various input and output connections for the SCB_I2C_PDL Component. An asterisk (*) in the following list indicates that it may not be shown on the Component symbol for the conditions listed in the description of that I/O.

| Name | I/O Type | Description |
|---|---|---|
| clock* | Digital Input | Clock that operates this block.<br>The presence of this terminal varies depending on the **Enable Clock from terminal** parameter. |
| scl_trig* | Digital Output | This output terminal allows monitoring of the SCL state. The connection capabilies are limited by the trigger mux. The typical connection is to TCPWM, which can be used to monitor the SCL low timeout.<br>The presence of this terminal varies depending on the **Enable SCL trigger output** parameter. |
| scl_b* | Digital Bidirectional | Serial clock (SCL) is the master-generated I²C clock. 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/NAK the latest data or address. The pin connected to scl typically should be configured as Open-Drain-Drives-Low.<br>The presence of this terminal varies depending on the **Show Terminals** parameter. |
| sda_b* | Digital Bidirectional | Serial data (SDA) is the I²C data signal. It is a bidirectional data signal used to transmit or receive all bus data. The pin connected to sda typically should be configured as Open-Drain-Drives-Low.<br>The presence of this terminal varies depending on the **Show Terminals** parameter. |

## Internal Pins Configuration

By default, the I²C pins are buried inside Component: I2C_1_scl and I2C_1_sda. These pins are buried because they use dedicated connections and are not routable as general purpose signals. For more information, refer to the I/O System section in the device Technical Reference Manual (TRM).

The preferred method to change pins configuration is to enable the **Show Terminals** option on the **Pins** tab and configure pins connected to the Component. Alternatively, the cy_gpio driver API can be used.

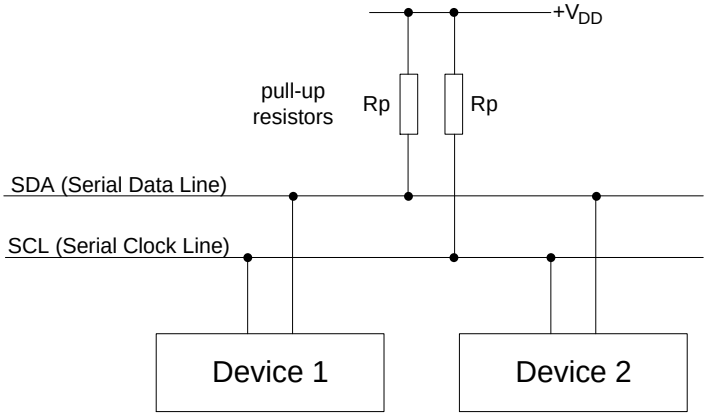
**Note** The instance name is not included in the Pin Names provided in the following tables:

| Pin Name | Direction | Drive Mode | Initial Drive State | Threshold | Slew Rate | Descriptiion |
|---|---|---|---|---|---|---|
| scl | Bidirectional | Open Drain Drives Low | High | CMOS | Fast | Serial clock (SCL) is the master-generated I²C clock.<br>This pins configuration requires connection of external pulls on the I²C bus. The other option is applying internal pull-ups setting pin Drive Mode to Resistive Pull-up. |
| sda | Bidirectional | Open Drain Drives Low | High | CMOS | Fast | Serial data (SDA) is the I²C data pin.<br>This pins configuration requires connection of external pulls on the I²C bus. The other option is applying internal pull-ups setting pin Drive Mode to Resistive Pull-up. |

The **Input threshold** level CMOS should be used for the vast majority of application connections. The **Output slew rate** is applied for whole port and set to Fast. The other Input and Output pin's parameters are set to default. Refer to pin Component datasheet for more information about parameters values.

## External Electrical Connections

As shown in the following figure, the $I^2C$ bus requires external pull-up resistors. The pull-up resistors ($R_P$) are primarily determined by the supply voltage, bus speed, and bus capacitance. For detailed information on how to calculate the optimum pull-up resistor value for your design, Cypress recommends using the I2C-bus specification and user manual.




For most designs, the default values shown in the following table provide excellent performance without any calculations. The default values were chosen to use standard resistor values between the minimum and maximum limits.

| Standard Mode (0 – 100 kbps) | Fast Mode (0 – 400 kbps) | Fast Mode Plus (0 – 1000 kbps) | Units |
|---|---|---|---|
| 4.7 k, 5% | 1.74 k, 1% | 620, 5% | Ω |

These values work for designs with 1.8 V to 5.0V $V_{DD}$, less than 200 pF bus capacitance ($C_B$), up to 25 µA of total input leakage ($I_{IL}$), up to 0.4 V output voltage level ($V_{OL}$), and a max $V_{IH}$ of 0.7 * $V_{DD}$.

Standard Mode and Fast Mode can use either GPIO [2] or GPIO_OVT PSoC pins. Fast Mode Plus requires use of GPIO_OVT pins to meet the $V_{OL}$ spec at 20 mA. Calculation of custom pull-up resistor values is required if your design does not meet the default assumptions, you use series resistors ($R_S$) to limit injected noise, or you want to maximize the resistor value for low power consumption.

Calculation of the ideal pull-up resistor value involves finding a value between the limits set by three equations detailed in the NXP $I^2C$ specification. These equations are:

Equation 1: $R_{PMIN} = (V_{DD}(max) – V_{OL}(max)) / I_{OL}(min)$

Equation 2: $R_{PMAX} = T_R(max) / 0.8473 \times C_B(max)$

Equation 3: $R_{PMAX} = V_{DD}(min) – (V_{IH}(min) + V_{NH}(min)) / I_{IH}(max)$

**Equation parameters:**

- $V_{DD}$ = Nominal supply voltage for $I^2C$ bus
- $V_{OL}$ = Maximum output low voltage of bus devices.
- $I_{OL}$= Low level output current from $I^2C$ specification
- $T_R$ = Rise Time of bus from $I^2C$ specification
- $C_B$ = Capacitance of each bus line including pins and PCB traces
- $V_{IH}$ = Minimum high level input voltage of all bus devices
- $V_{NH}$ = Minimum high level input noise margin from $I^2C$ specification
- $I_{IH}$ = Total input leakage current of all devices on the bus

The supply voltage ($V_{DD}$) limits the minimum pull-up resistor value due to bus devices maximum low output voltage ($V_{OL}$) specifications. Lower pull-up resistance increases current through the pins and can therefore exceed the spec conditions of $V_{OH}$. Equation 1 is derived using Ohm's law to determine the minimum resistance that will still meet the $V_{OL}$ specification at 3 mA for standard and fast modes, and 20 mA for fast mode plus at the given $V_{DD}$.

Equation 2 determines the maximum pull-up resistance due to bus capacitance. Total bus capacitance is comprised of all pin, wire, and trace capacitance on the bus. The higher the bus capacitance the lower the pull-up resistance required to meet the specified bus speeds rise time due to RC delays. Choosing a pull-up resistance higher than allowed can result in failing timing requirements resulting in communication errors. Most designs with five of fewer $I^2C$ devices and up to 20 centimeters of bus trace length have less than 100 pF of bus capacitance.

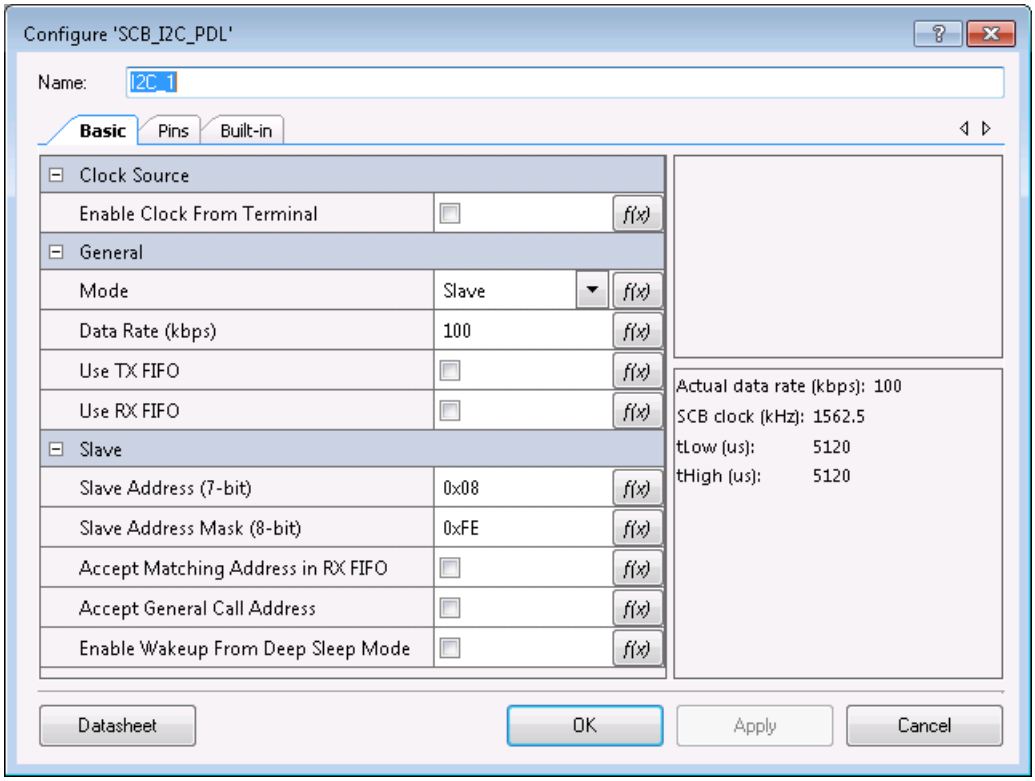
A secondary effect that limits the maximum pull-up resistor value is total bus leakage calculated in Equation 3. The primary source of leakage is I/O pins connected to the bus. If leakage is too high, the pull-ups will have difficulty maintaining an acceptable $V_{IH}$ level causing communication errors. Most designs with five or fewer $I^2C$ devices on the bus have less than 10 µA of total leakage current.

# Component Parameters

The SCB_I2C_PDL Component Configure dialog allows you to edit the configuration parameters for the Component instance.

## Basic Tab

This tab contains the Component parameters used in the general peripheral initialization settings.



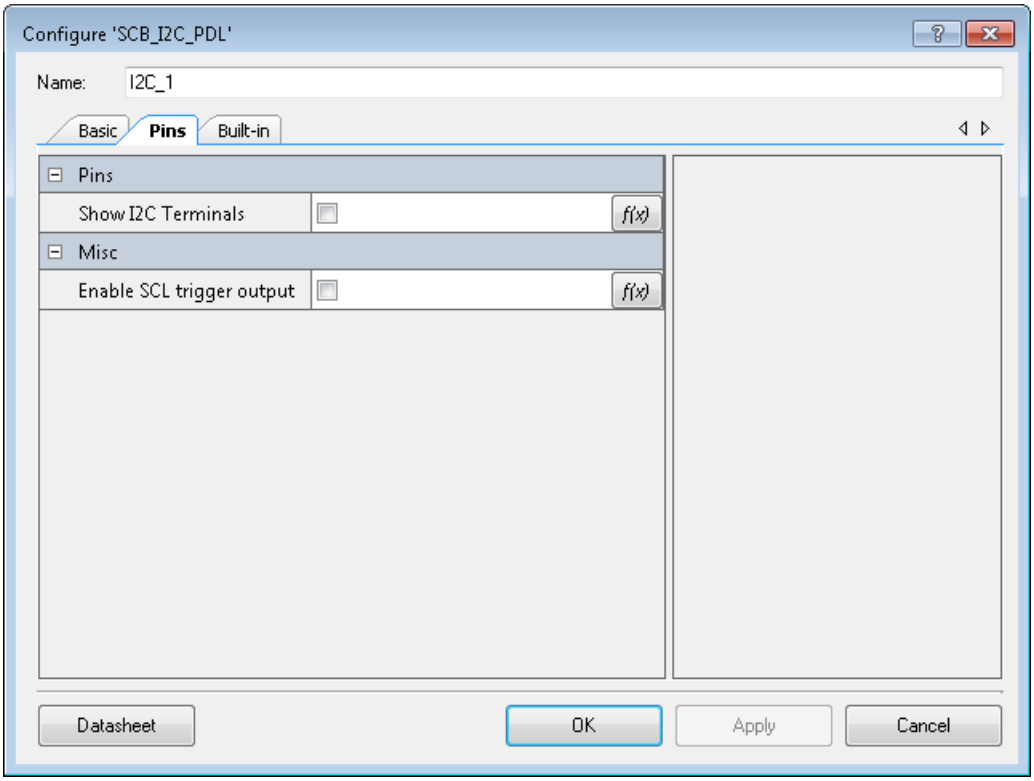

| Parameter Name | Description |
|---|---|
| Enable Clock from Terminal | This parameter allows choosing between an internally configured clock (by the Component) or an externally configured clock (by the user) for Component operation. |
| Mode | This parameter defines the I²C operation mode as: slave, master, or master-slave. |
| Data Rate (kbps) | This parameter specifies the data rate in kbps. The actual data rate may differ from the selected data rate due to the available clock frequency and Component settings. The standard data rates are 100 (default), 400, and 1000 kbps. The range: 1-1000 kbps. |
| Actual Data Rate (kbps) | The actual data rate displays the data rate at which the Component will operate with current settings. The factors that affect the actual data rate calculation are: the accuracy of the Component clock (internal or external) and SCL Low and High Phase (only for the master mode). |

| Parameter Name | Description |
|---|---|
| Use RX FIFO | The SCB provides an RX FIFO in hardware (consult the selected device datasheet to get actual FIFO size). The useRxFifo field defines how the driver firmware reads data from the RX FIFO:<br>• If this option is enabled, the hardware is configured to automatically ACK incoming data, and interrupt is enabled to take data out of the RX FIFO when it has some number of bytes (typically, when it is half full).<br>• If this option is disabled, the interrupt is enabled to take data out of the RX FIFO when a byte is available. Also, hardware does not automatically ACK the data, firmware must tell the hardware to ACK or NACK the byte (so each byte requires interrupt processing).<br>Typically, this option should be enabled to configure hardware to automatically ACK incoming data. Otherwise hardware might not get command to ACK or NACK a byte fast enough and clock stretching is applied (the transaction is delayed) until command is set. Also the number of interrupts to process the transaction is significantly reduced because a number of bytes is handled at once when the useRxFifo is enabled. However, before enabling this option the drawback must be analyzed:<br>• For the master mode, the drawback is that the master may receive more data than desired due to the interrupt latency. An interrupt fires when the second-to-last byte has been received, this interrupt tells the hardware to stop receiving data. If the latency of this interrupt is longer than 1 transfer of the byte on the I2C bus, then the hardware automatically ACKs the following bytes until the interrupt is serviced or the RX FIFO becomes full.<br>• For the slave mode, the drawback is that the slave only NACKs the master when the RX FIFO becomes full, NOT when the slave write firmware buffer becomes full.<br>**Note** Any extra bytes received in the RX FIFO beyond the size of the firmware buffer are dropped.<br>This parameter is not available if the **Accept Matching Address** in RX FIFO parameter is enabled. |
| Use TX FIFO | The SCB provides a TX FIFO in hardware (consult the selected device datasheet to get actual FIFO size). The useTxFifo option defines how the driver firmware loads data into the TX FIFO:<br>• If this option is enabled, the TX FIFO is fully loaded with data and the interrupt is enabled to keep the TX FIFO loaded until end of transaction.<br>• If this option is disabled, a single byte is loaded in the TX FIFO and the interrupt enabled to load next byte when the TX FIFO becomes empty (so each byte requires interrupt processing).<br>Typically, this option should be enabled to keep the TX FIFO loaded with data and reduce the probability of clock stretching. When there is no data to transfer clock stretching is applied (the transaction is delayed) until data is loaded. The number of interrupts to execute the transaction is reduced as well.<br>The drawback of enabling useTxFifo is that the abort operation clears the TX FIFO. The TX FIFO clear operation also clears the shift register so that the shifter may be cleared in the middle of a byte transfer, corrupting it. The remaining bits to transfer within the corrupted byte are complemented with 1s. If this is an issue, then do not enable this option. |

| Parameter Name | Description |
|---|---|
| Enable Manual SCL Control | This parameter specifies the method of calculating the SCL low-phase and high-phase duty cycle as automatic or manual (only applicable for the master modes). |
| SCL Low Phase (SCB Clocks) | This parameter defines how many Component clocks are used to generate the SCL low phase (only applicable for master modes). The range: 7-16. |
| SCL High Phase (SCB Clocks) | This parameter defines how many Component clocks are used to generate the SCL high phase (only applicable for master modes). The range: 5-16. |
| Slave Address (7-bit) | This parameter specifies the 7-bit right justified slave address. The range: 0x08-0x78. |
| Slave Address Mask (8-bit) | This parameter specifies the slave address mask. The range: 0x00-0xFE.<br>• Bit value 0 – excludes the bit from the address comparison.<br>• Bit value 1 – the bit needs to match with the corresponding bit of the $I^2C$ slave address. |
| Accept Matching Address in RX FIFO | This parameter determines whether to accept the match slave address in the RX FIFO or not. This feature is useful when more than one address support is required. The user has to register the callback function to handle the accepted addresses. |
| Accept General Call Address | This parameter specifies whether to accept the general call address. The general call address is ACKed when accepted and NAKed otherwise. The user has to register the callback function to handle the general call address. |
| Enable Wakeup from Deep Sleep Mode | This parameter enables the Component to wake the system from Deep Sleep when a slave address match occurs (only applicable for slave mode). |

### Pins Tab

This tab contains the Interrupt configuration settings.




| Parameter Name | Description |
|---|---|
| Show I2C Terminals | This parameter removes internal pins and exposes signals to terminals. The exposed terminals must be connected to Pins Components. |
| Enable SCL trigger output | This parameter enables scl_trig output. This allows connecting the SCL line to the trigger mux so that it can be monitored by a TCPWM. |

# Application Programming Interface

The Application Programming Interface (API) is provided by the cy_scb driver module from the PDL. The driver is copied into the “pdl\drivers\peripheral\scb\” directory of the application project after a successful build.

Refer to the PDL documentation for a detailed description of the complete API. To access this document, right-click on the Component symbol on the schematic and choose the “**Open PDL Documentation…**” option in the drop-down menu.

The Component generates the configuration structures and base address described in the Global Variables and Preprocessor Macros sections. Pass the generated data structure and the

base address to the associated cy_scb driver function in the application initialization code to configure the peripheral. Once the peripheral is initialized, the application code can perform run-time changes by referencing the provided base address in the driver API functions.

By default, PSoC Creator assigns the instance name I2C_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.

## Global Variables

The SCB_I2C_PDL Component populates the following peripheral initialization data structure(s). The generated code is placed in C source and header files that are named after the instance of the Component (e.g., *I2C_1.c*). Each variable is also prefixed with the instance name of the Component.

**cy_stc_scb_i2c_config_t const I2C_1_config**

The instance-specific configuration structure. The pointer to this structure should be passed to Cy_SCB_I2C_Init function to initialize Component with GUI selected settings.

**cy_stc_scb_i2c_context_t I2C_1_context**

The instance-specific context structure. It is used for internal configuration and data keeping for the I²C driver. Do not modify anything in this structure. If only Master Low-Level functions are used, this is not needed.

## Preprocessor Macros

The SCB_I2C_PDL Component generates the following preprocessor macro(s). Note that each macro is prefixed with the instance name of the Component (e.g. “I2C_1”).

**#define I2C_1_HW ((CySCB_Type *) I2C_1_SCB__HW)**

The pointer to the base address of the SCB instance

**#define I2C_1_DATA_RATE_HZ**

The desired data rate in Hz if **Clock from Terminal** parameter is false and actual data rate otherwise

**#define I2C_1_CLK_FREQ_HZ**

The frequency of the clock used by the Component in Hz

**#define I2C_1_LOW_PHASE_DUTY_CYCLE**

The number of Component clocks used by the master to generate the SCL low phase. This number is calculated by the GUI based on the selected data rate.

**#define I2C_1_HIGH_PHASE_DUTY_CYCLE**

The number of Component clocks used by the master to generate the SCL high phase. This number is calculated by the GUI based on the selected data rate.

## Data in RAM

The generated data may be placed in flash memory (const) or RAM. The former is the more memory-efficient choice if you do not wish to modify the configuration data at run-time. Under the **Built-In** tab of the Configure dialog, set the parameter CONST_CONFIG to make your selection. The default option is to place the data in flash.

## Interrupt Service Routine

The interrupt service routine (ISR) is mandatory for the SCB_I2C_PDL Component; therefore, an Interrupt Component is placed inside it. The Cy_SCB_I2C_Interrupt function (from the cy_scb driver) implements the ISR functionality. You must call the Cy_SCB_I2C_Interrupt function inside the ISR and enable the interrupt controller to trigger the corresponding interrupt. Refer to the code example in the Quick Start section.

**EXCEPTION** When the Component is configured for Master operation and only I²C Low-Level communication API of the cy_scb driver are used, then interrupt handling is not needed.

All slave functions, except Cy_SCB_I2C_SlaveAbortRead and Cy_SCB_I2C_SlaveAbortWrite, are not interrupt protected. To prevent a race condition, they should be protected from I²C interruption in places where they are called. For example:

```
/* Disable I2C_1 Interrupt: protect code from interruption */
NVIC_DisableIRQ(I2C_1_SCB_IRQ_cfg.intrSrc);

/* Check if write transfer completed */
if (0UL != (CY_SCB_I2C_SLAVE_WR_CMPLT & Cy_SCB_I2C_SlaveGetStatus(I2C_1_HW,
&I2C_1_context)))
{
    /* Process or copy the I2C slave write buffer here */

    /* Configure buffer for the next write */
    Cy_SCB_I2C_SlaveConfigWriteBuf(I2C_1_HW, buffer, BUFFER_SIZE, &I2C_1_context);

    /* Clear slave write status to capture following updates */
    Cy_SCB_I2C_SlaveClearWriteStatus(I2C_1_HW, &I2C_1_context);
}

/* Enable I2C_1 interrupt */
NVIC_EnableIRQ(I2C_1_SCB_IRQ_cfg.intrSrc);
```

Refer to the cy_scb driver documentation for more information about I²C master High-Level and Low-Level communication functions.

## Code Examples and Application Notes

This section lists the projects that demonstrate the use of the Component.

### Code Examples

PSoC Creator provides access to code examples in the Code Example 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 "Code Example" topic in the PSoC Creator Help for more information.

There are also numerous code examples that include schematics and example code available online at the Cypress Code Examples web page.

### Application Notes

Cypress provides a number of application notes describing how PSoC can be integrated into your design. You can access the Cypress Application Notes search web page at www.cypress.com/appnotes.

# Functional Description

## Data Rate Configuration

The Component must meet the data rate requirement of the connected $I^2C$ bus. For master mode, this means the master data rate cannot be faster than the slowest slave in the system. For slave mode, this means the slave cannot be slower than the fastest master in the system.

### Slave Mode

The frequency of the connected clock source is the only parameter used in determining the maximum data rate at which the slave can operate. The connected clock is the clock that runs the SCB hardware, not SCL. The frequency of the connected clock source must be fast enough to provide enough oversampling of the SCL and SDA signals and ensure that all $I^2C$ specifications are met.

The Component provides the following methods to configure the **Data Rate**:

- Set the desired **Data Rate**. This option uses a clock internal to the Component (this clock still uses clock divider resources). Based on the data rate, the Component asks PSoC Creator to create a clock with a frequency to satisfy data rate requirements.

- Connect a user-configurable Clock Component to the terminal. This option is controlled by the **Enable Clock from Terminal** parameter, which must be enabled. In this mode, ensure the connected clock frequency is fast enough to support your system data rate.

  For more information about $I^2C$ data rate configuration, refer to the Inter Integrated Circuit ($I^2C$) Oversampling and Bit Rate sub-section in the device TRM.

Regardless of the chosen method, the Component will display the **Actual data rate**. This is the maximum data rate at which the slave can operate. If the system data rate is faster than the displayed actual data rate, proper $I^2C$ operation is no longer guaranteed.

### Master Modes (includes Master-Slave)

The data rate is determined by the connected clock source and the oversampling factor. These two factors are used to set the frequency of the SCL. One SCL period is equal to the period of the connected clock multiplied by the oversampling factor. The oversampling factor is divided into low and high phases to enable independent control of the low and high SCL duration. The low and high oversampling factor can be configured independently, but their sum must be equal to the overall oversampling. To ensure that the master meets all $I^2C$ specifications, the connected clock frequency and oversampling factor must be within a specified range.

The Component provides the following methods to configure the **Data Rate**:

- Set the desired **Data Rate**. This option uses a clock internal to the Component (this clock still uses clock divider resources). Based on the data rate, the Component asks PSoC Creator to create a clock with a frequency in the specified range. When the available clock frequency is returned, the SCL low and high phases are calculated to meet the desired **Data Rate**.

- Connect a user-configurable Clock Component to the terminal. This option is controlled by the **Enable Clock from Terminal** parameter, which must be enabled. In addition, the **Enable Manual SCL Control** parameter must be enabled to configure **SCL Low Phase** and **SCL High Phase**. This method provides you with full control of the data rate configuration.

  For more information about $I^2C$ data rate configuration, refer to the Inter Integrated Circuit ($I^2C$) Oversampling and Bit Rate sub-section in the device TRM.

Regardless of the chosen method, the Component will display the **Actual data rate**.

**Note** Actual data rate might differ from the observed data rate on the bus due to the $t_R$ and $t_F$ time.

### Clock Selection

The SCB_I2C_PDL Component supports two clock select options: internal and external.

- Internal means that the Component is responsible for clock configuration. It requests that the system provides the clock frequency necessary to operate with the selected the data rate.

  **Note** When the Component is configured as master or master-slave, the input digital filters are enabled instead of analog filters to operate in fast plus mode. The input analog filters are used for any other Component configurations.

- External means that the Component provides a clock terminal to connect a Clock Component. You must then configure the Clock Component appropriately.

For more information about I²C clock configuration, refer to the Inter Integrated Circuit (I²C) Oversampling and Bit Rate sub-section in the device TRM.

### Low-Power Modes

The SCB_I2C_PDL Component requires specific processing to support Deep Sleep and Hibernate modes. The cy_scb driver module provides callback functions to handle power mode transition. These functions must be registered using the cy_syspm driver before entering Deep Sleep or Hibernate mode appropriately. Refer to the Low Power Support section of the cy_scb driver, or the System Power Management section of the PDL Documentation for more details about callback registration.

The Slave is capable of waking up the device from Deep Sleep mode on address match. To enable this functionality, select the Enable Wakeup from Deep Sleep Mode parameter and register the Deep Sleep callback.

The Master is **not** able to wake up the device from Deep Sleep mode.

## Industry Standards

### I²C-bus specification

The SCB_I2C_PDL Component is compatible [2] with I²C Standard-mode, Fast-mode, and Fast-mode Plus devices as defined in the *I²C-bus specification and user manual* [3].

---

2 PSoC 6 pins are not completely compliant with the I²C specification except GPIO_OVT pins. For detailed information, refer to the selected device datasheet.

3 The UM10204 I2C-bus specification and user manual Rev. 6 – 4 April 2014 is supported.

### 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. Refer to PSoC Creator Help > Building a PSoC Creator Project > Generated Files (PSoC 6) for information on MISRA compliance and deviations for files generated by PSoC Creator.

The SCB_I2C_PDL Component does not have any specific deviations.

This Component uses firmware drivers from the cy_scb, cy_sysint, and cy_gpio PDL modules. For information on their MISRA compliance and specific deviations, refer to the PDL documentation.

This Component has the following embedded Components: clock, interrupt and pins. Refer to the corresponding Component datasheets for information on their MISRA compliance and specific deviations.

## Registers

Refer to the Serial Communication Block Registers section in the device TRM.

## Resources

The SCB_I2C_PDL Component uses a single SCB peripheral block configured for $I^2C$ operation.

## DC and AC Electrical Characteristics

**Note** Final characterization data for PSoC 6 devices is not available at this time. Once the data is available, the Component datasheet will be updated on the Cypress web site.

# Component Changes

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

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 2.0.b | Updated the datasheet. | Added Low-Power Modes section. |
| | Fixed the I²C pins configuration sequence so that it does not cause strong pull-up before the I2C_Start() function is called. | Strong pull-up can corrupt an ongoing I²C transaction issued by another I²C device on the bus. |
| 2.0.a | Minor datasheet edits. | |
| 2.0 | Updated the underlying version of PDL driver. | |
| | Added scl_trig signal to allow SCL line monitoring. | |
| | Fixed initialization of useRxFifo field of the I2C_1_config. | |
| | Datasheet updates. | |
| 1.0.a | Datasheet updates. | |
| 1.0 | Initial Version | |

© Cypress Semiconductor Corporation, 2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical Components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical Component is any Component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.