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.

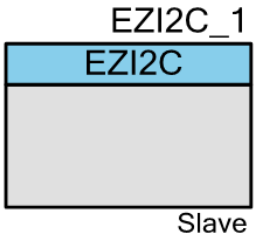
Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

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.

EZI2C (SCB_EZI2C_PDL)

2.0

Features



- Industry-Standard I^2C bus interface
- Supports data rates of 100/400/1000 kbps
- Emulates common I^2C EEPROM Interface
- Acts like dual port memory between the external master and your code
- Hardware Address Match
- Supports two hardware addresses with separate buffers
- Wake from Deep Sleep on address match
- Simple to setup and use. Once setup no need to call EZI2C API in run time

General Description

The SCB_EZI2C_PDL Component is a unique implementation of an I^2C slave in that all communication between the master and slave is handled in the Interrupt Service Routine (ISR). It requires no interaction with the main program flow. The interface appears as shared memory between the master and slave.

The SCB_EZI2C_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_EZI2C_PDL Component

Use the SCB_EZI2C_PDL Component when you want a shared memory model between the I^2C slave and I^2C master. You may define the EZI2C slave buffers as any variable, array, or structure in your code without worrying about the I^2C protocol. The I^2C master may view any of the variables in this buffer and modify them.

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 I²C, SPI, and UART interfaces.

Quick Start

1. Drag a EZI2C (SCB) Component from the Component Catalog *Cypress/Communications/I2C* folder onto your schematic (placed instance takes the name EZI2C_1).
2. Double-click to open the Configure dialog.
3. On the **Basic** tab, select the **Data rate** at which the I²C interface is expected to communicate and the **Primary Slave Address**.
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, generate configuration data, and allocate context for the EZI2C_1 instance.
6. In the *main.c* file, initialize the peripheral and start the application:

```
/* Implement ISR for EZI2C_1 */
void EZI2C_1_Isr(void)
{
    Cy_SCB_EZI2C_Interrupt(EZI2C_1_HW, &EZI2C_1_context);
}

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

/* Initialize SCB for EZI2C operation with GUI selected settings */
(void) Cy_SCB_EZI2C_Init(EZI2C_1_HW, &EZI2C_1_config, &EZI2C_1_context);

/* Hook interrupt service routine and enable interrupt */
Cy_SysInt_Init(&EZI2C_1_SCB_IRQ_cfg, &EZI2C_1_Isr);
NVIC_EnableIRQ(EZI2C_1_SCB_IRQ_cfg.intrSrc);

/* Configure buffer for communication with master */
Cy_SCB_EZI2C_SetBuffer1(EZI2C_1_HW, buffer, BUFFER_SIZE, BUFFER_SIZE,
                        &EZI2C_1_context);

/* Enable EZI2C */
Cy_SCB_EZI2C_Enable(EZI2C_1_HW);
```

7. Build and program the device.

Input/Output Connections

This section describes the various input and output connections for the SCB_EZI2C_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.

Terminal Name	I/O Type	Description
clock*	Digital Input	Clock that operates this block. The presence of this terminal varies depending on the **Enable Clock from Terminal** parameter.
scl_b*	Digital Bidirectional	Serial clock (SCL) is the master-generated I^2C 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. The presence of this terminal varies depending on the **Show EZI2C Terminals** parameter.
sda_b*	Digital Bidirectional	Serial data (SDA) is the I^2C 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. The presence of this terminal varies depending on the **Show EZI2C Terminals** parameter.

Internal Pins Configuration

By default, the I^2C pins are buried inside Component: EZI2C_1_scl and EZI2C_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 **Show EZI2C 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 Name provided in the following table.

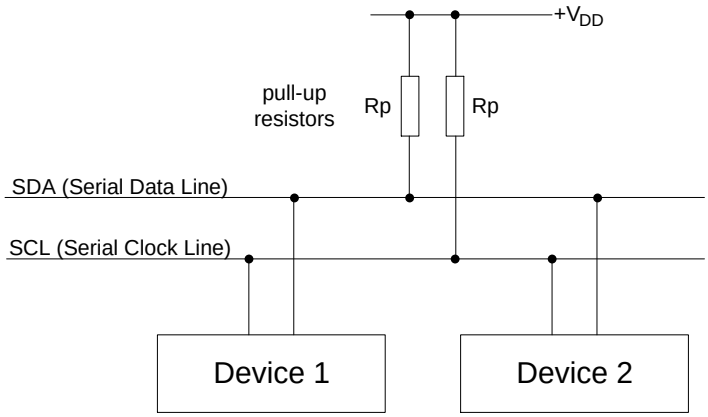
Pin Name	Direction	Drive Mode	Initial Drive State	Threshold	Slew Rate	Descriptiion
scl	Bidirectional	Open Drain Drives Low	High	CMOS	Fast	Serial clock (SCL) is the master-generated I^2C clock. This pin configuration requires connection of external pulls on the I^2C bus. The other option is applying internal pull-ups setting pin Drive Mode to Resistive Pull-up.
sda	Bidirectional	Open Drain Drives Low	High	CMOS	Fast	Serial data (SDA) is the I^2C data pin. This pin configuration requires connection of external pulls on the I^2C 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 I²C 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 we recommend 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 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 I²C 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.

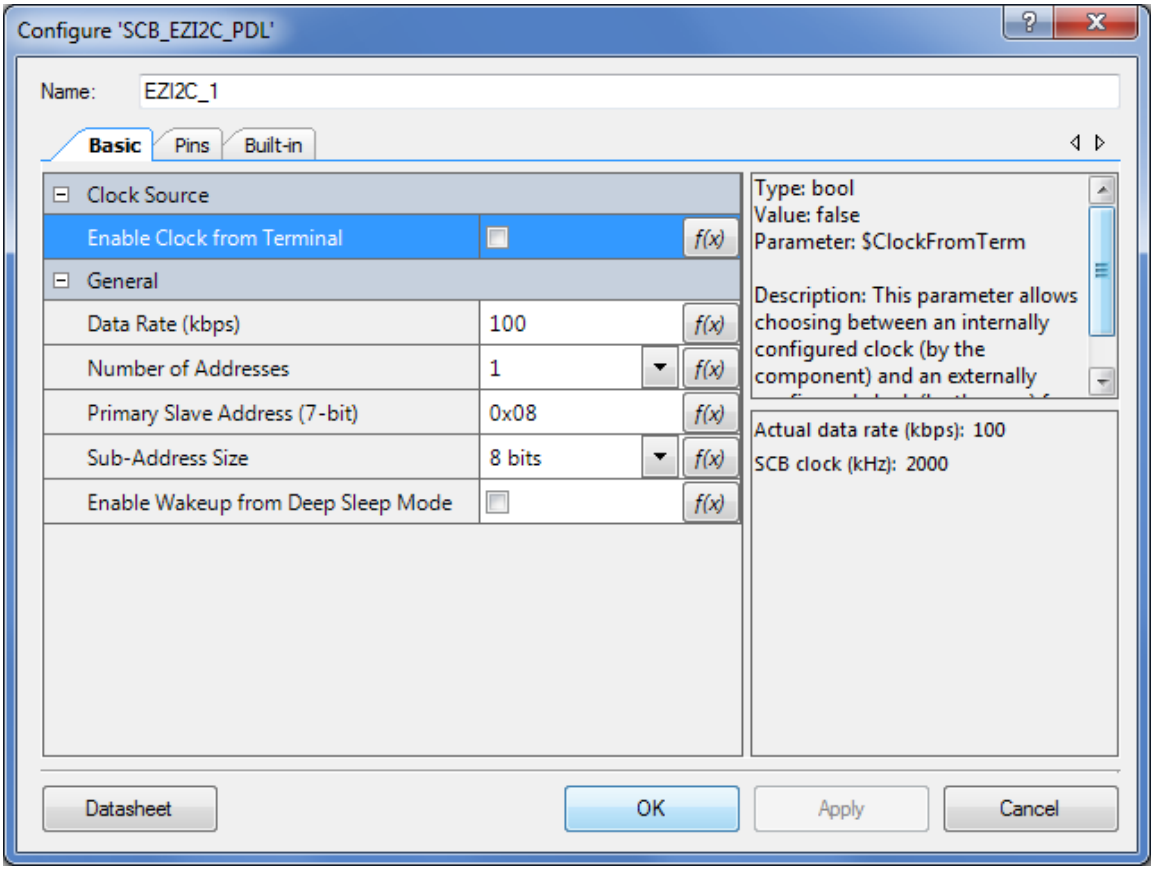
A secondary effect that limits the maximum pull-up resistor value is total bus leakage calculated in Equation 3. The primary source of leakage is I/O pins connected to the bus. If leakage is too high, the pull-ups will have difficulty maintaining an acceptable V_{IH} level causing communication errors. 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_EZI2C_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.

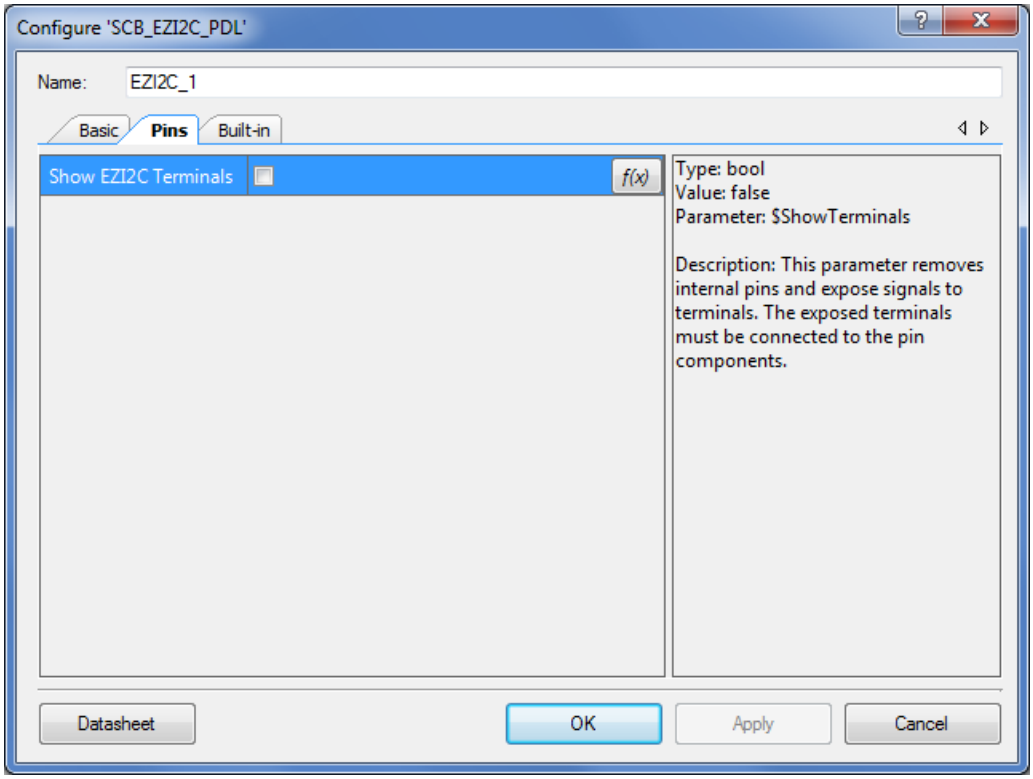


Parameter Name	Description
Enable Clock from Terminal	This parameter allows choosing between an internally configured clock (by the Component) or an externally configured clock (by the user) for Component operation.
Data Rate (kbps)	This parameter specifies the data rate in kbps. The actual data rate 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 factor that affect the actual data rate calculation is: the nominal frequency of the Component clock (internal or external).
Number of Addresses	This parameter specifies whether 1 or 2 independent I^2C slave addresses are recognized.

Parameter Name	Description
Primary Slave Address (7-bit)	This parameter specifies the 7-bit right justified primary slave address.
Secondary Slave Address (7-bit)	This parameter specifies the 7-bit right justified secondary slave address.
Sub-Address Size	This option determines what range in the slave buffer can be accessed by the master. For a sub-address size of 8 bits, the master can only access a buffer in the range between 0 and 255. Whereas for 16 bits, the master can access a buffer in the range between 0 and 65,535.
Enable Wakeup from Deep Sleep Mode	This parameter enables the Component to wake the system from Deep Sleep when a slave address match occurs.

Pins Tab

This tab contains the Interrupt configuration settings.



Parameter Name	Description
Show EZI2C Terminals	This parameter removes internal pins and expose signals to terminals. The exposed terminals must be connected to the pin components.

Application Programming Interface

The Application Programming Interface (API) is provided 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 EZI2C_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_EZI2C_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., *EZI2C_1.c*). Each variable is also prefixed with the instance name of the Component.

cy_stc_scb_ezi2c_config_t const EZI2C_1_config

The instance-specific configuration structure. The pointer to this structure should be passed to Cy_SCB_EZI2C_Init function to initialize Component with GUI selected settings.

cy_stc_scb_ezi2c_context_t EZI2C_1_context

The instance-specific context structure. It is used while the driver operates in internal configuration and data keeping for the EZI2C. Do not modify anything in this structure.

Preprocessor Macros

The SCB_EZI2C_PDL Component generates the following preprocessor macro(s). Note that each macro is prefixed with the instance name of the Component (e.g. "EZI2C_1").

#define EZI2C_1_HW ((CySCB_Type *) EZI2C_1_SCB__HW)

The pointer to the base address of the SCB instance.

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_EZI2C_PDL Component; therefore, an Interrupt Component is placed inside it. The cy_scb driver function Cy_SCB_EZI2C_Interrupt implements the ISR functionality. You must call the Cy_SCB_EZI2C_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.

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 Find 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 "Find 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

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

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.

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

- Set the desired **Data Rate**. This option uses a clock that is 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 to the Component. This option is controlled by the **Enable Clock from Terminal** parameter, which must be enabled. In this mode, you must 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.

Clock Selection

The SCB_EZI2C_PDL Component provides the **Enable Clock from Terminal** parameter, which allows choosing between an internally configured clock (by the Component) or an externally configured clock for Component operation:

- Internally configured means that the Component is responsible for clock configuration. It requests the system to provide the clock frequency necessary to operate with selected the data rate.
- Externally configured 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 I^2C slave clock configuration, refer to the Inter Integrated Circuit (I^2C) Oversampling and Bit Rate sub-section in the device TRM.

Low-Power Modes

The SCB_EZI2C_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 EZI2C 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.

Industry Standards

I²C-bus specification

The SCB_EZI2C_PDL Component is compatible [1] with I²C Standard-mode, Fast-mode, and Fast-mode Plus devices as defined in *I²C-bus specification and user manual* [2].

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_EZI2C_PDL Component does not have any specific deviations.

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

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.

---

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

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

Registers

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

Resources

The SCB_EZI2C_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 I^2C pins configuration sequence so that it does not cause strong pull-up before the EZI2C_Start() function is called.	Strong pull-up can corrupt an ongoing I^2C transaction issued by another I^2C device on the bus.
2.0.a	Minor datasheet edits.	
2.0	Updated the underlying version of PDL driver.	
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.