**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.

# CE195362 – PSoC 4 EZI2C Slave with Serial Communication Block (SCB)

## Objective

This code example demonstrates the implementation of an EZI2C Slave using the PSoC® Creator™ SCB Component on a PSoC 4 device. It also demonstrates how to control the color and intensity of an RGB LED using Timer Counter Pulse Width Modulator (TCPWM) Components.

## Overview

This code example implements an I²C Slave using a Serial Communication Block (SCB) Component (configured as EZI2C), which receives the data required to control an RGB LED from an I²C Master. In this example, a host PC running the Cypress Bridge Control Panel (BCP) software is used as an I²C Master. RGB LED control is implemented using three TCPWM Components (configured as PWM). The color and intensity of the RGB LED is controlled by changing the duty cycle of the PWM signals.

## Requirements

**Tool:** PSoC Creator 4.2

**Programming Language:** C (Arm® GCC 5.4.1 and Arm MDK 5.22)

**Associated Parts:** PSoC 4 parts

**Related Hardware:** CY8CKIT-041-40XX, CY8CKIT-041-41XX, CY8CKIT-042, CY8CKIT-042-BLE, CY8CKIT-042-BLE-A, CY8CKIT-044, CY8CKIT-046, CY8CKIT-048, CY8CKIT-149

## Hardware Setup

By default, this example project is configured to run on the CY8CKIT-042 development kit from Cypress Semiconductor. The project can be migrated to any supported kit by changing the target device. Open the **Device Selector** from the project's context menu. Table 1 lists supported kits and corresponding devices.

This example uses the kit's default configuration. Refer to the kit guide to ensure that the kit is configured correctly.

Table 1. Supported Kits and Devices

| Development Kit | Series | Device |
|---|---|---|
| CY8CKIT-041-40XX | PSoC 4000S | CY8C4045AZI-S413 |
| CY8CKIT-041-41XX | PSoC 4100S | CY8C4146AZI-S433 |
| CY8CKIT-042 | PSoC 4200 | CY8C4245AXI-483 |
| CY8CKIT-042-BLE | PSoC 4200 BLE | CY8C4247LQI-BL483 |
| CY8CKIT-042-BLE-A | PSoC 4200 BLE | CY8C4248LQI-BL483 |
| CY8CKIT-044 | PSoC 4200M | CY8C4247AZI-M485 |
| CY8CKIT-046 | PSoC 4200L | CY8C4248BZI-L489 |
| CY8CKIT-048 | PSoC Analog Coprocessor | CY8C4A45LQI-483 |
| CY8CKIT-149 | PSoC 4100S Plus | CY8C4147AZI-S475 |

Pin assignments for supported kits are provided in Table 2. For these kits, the project includes control files to automatically assign pins with respect to the kit hardware connections during project build. To change pin assignments, override control file selections in the **Pin Editor** of the **Design Wide Resources** by selecting the new port or pin number.

Table 2. Pin Assignments

| Development Kit | Pin Assignment | | | | |
|---|---|---|---|---|---|
| | \I2C:scl\ | \I2C:sda\ | LED_Red | LED_Green | LED_Blue |
| CY8CKIT-041-40XX | P3[0] | P3[1] | P3[4] | P2[6] | P3[6] |
| CY8CKIT-041-41XX | P3[0] | P3[1] | P3[4] | P2[6] | P3[6] |
| CY8CKIT-042 | P3[0] | P3[1] | P1[6] | P0[2] | P0[3] |
| CY8CKIT-042-BLE | P3[5] | P3[4] | P2[6] | P3[6] | P3[7] |
| CY8CKIT-042-BLE-A | P3[5] | P3[4] | P2[6] | P3[6] | P3[7] |
| CY8CKIT-044 | P4[0] | P4[1] | P0[6] | P2[6] | P6[5] |
| CY8CKIT-046 | P4[0] | P4[1] | P5[2] | P5[3] | P5[4] |
| CY8CKIT-048 | P4[0] | P4[1] | P1[4] | P2[6] | P1[6] |
| CY8CKIT-149 | P3[0] | P3[1] | P5[2] | P5[5] | P5[7] |

**Note**: CY8CKIT-149 does not have an RGB LED. Instead, this example controls LED11, LED12, and LED13, which are all green.

## Software Setup

For this code example, you need the Bridge Control Panel software, which is installed with PSoC Creator.

## Operation

1. Plug your kit board into your computer's USB port.
2. For PSoC 4000S, 4100S, or Analog Coprocessor parts, right click project **EZI2C_Slave_SCB_4S_Analog_Coprocessor** and select **Set As Active Project**.

   For all other supported PSoC 4 devices, right click project **EZI2C_Slave_SCB** and select **Set As Active Project**.
3. Build the project and program it into the PSoC 4 device. Choose **Debug** > **Program**. For more information on device programming, see the PSoC Creator Help.
4. Confirm that the RGB LED is green.
5. Open Bridge Control Panel (**Start** > **All Programs** > **Cypress** > **Bridge Control Panel**<**Version**> > **Bridge Control Panel** <**Version**>).
6. Select the KitProg device in **Connected I2C/SPI/RX8 Ports**. Make sure that the selected protocol is $I^2C$ (Figure 1).
7. Go to **Tools** > **Protocol Configuration**, and in the **I2C** tab, select **I2C Speed** as **100 kHz**.
8. Press the **List** button and confirm that the EZI2C Slave device with the address 0x08 (7 bits) is available for communication.
9. In the **Editor** tab of the BCP, type the command to send the RGB LED control data, and then click **Send**. Observe that the RGB LED turns ON with the specified color and intensity.

   Each command requires writing and reading specific locations of the EZI2C Buffer. For more details on how this buffer is set up, see Design and Implementation.

   The packet format for writing to a Slave device from the BCP is shown below.

| Start for Write | Slave Address | Slave Buffer Index to Start Write | Red LED TCPWM Compare Value | Green LED TCPWM Compare Value | Blue LED TCPWM Compare Value | Stop |
|---|---|---|---|---|---|---|
| w | (0x08) | (0x00) to (0x02) | (0x00) to (0xFF) | (0x00) to (0xFF) | (0x00) to (0xFF) | p |

For example, sending the command ‘w 08 00 00 00 00 p’ will turn the RGB LED OFF. The command ‘w 08 00 FF FF FF p’ will turn the RGB LED ON white with full intensity.

You can also control individual LEDs by writing the value to the specific Slave buffer location. For example, sending the command ‘w 08 02 7F p’ will turn the RGB LED ON with blue color and medium intensity.

You can check the number of write operations that are performed by reading index 0x03 of the Slave buffer. This can be done by setting the write index to the index of the counter, ‘w 08 03 p’, and then reading that index with ‘r 08 x p’.

The packet format for reading data from the Slave device is shown below.

| Start for Read | Slave Address | Red LED TCPWM Compare Value | Green LED TCPWM Compare Value | Blue LED TCPWM Compare Value | Number of Write Operations | Stop |
|---|---|---|---|---|---|---|
| r | (0x08) | x | x | x | x | p |

For example, sending the command ‘r 08 x x x p’ will return the last value written to each of the TCPWM Components as well as the number of write operations performed.

For more information on the syntax for Bridge Control Panel commands, go to the Bridge Control Panel help menu and click on the Communication tab in the Contents menu (**Bridge Control Panel** > **Help Contents** > **Communication**).

Figure 1. Bridge Control Panel I²C Master Setup

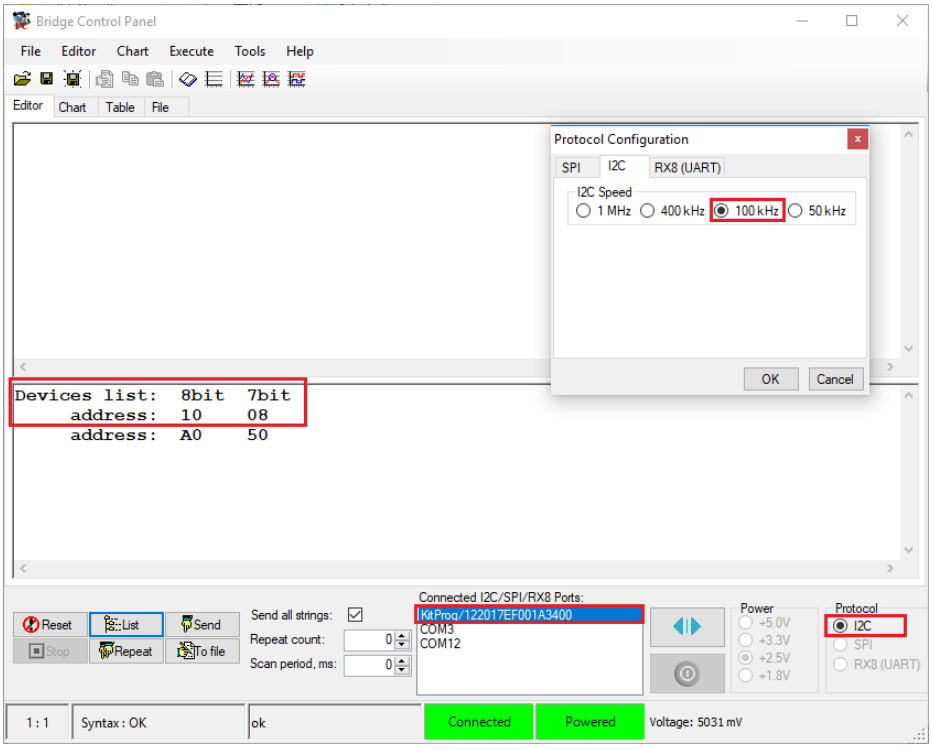
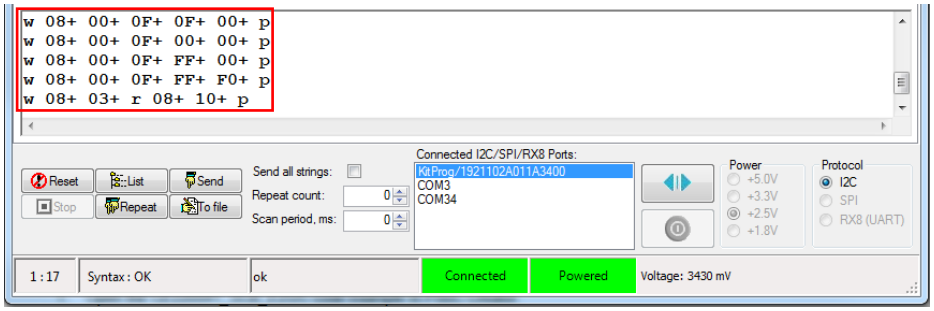


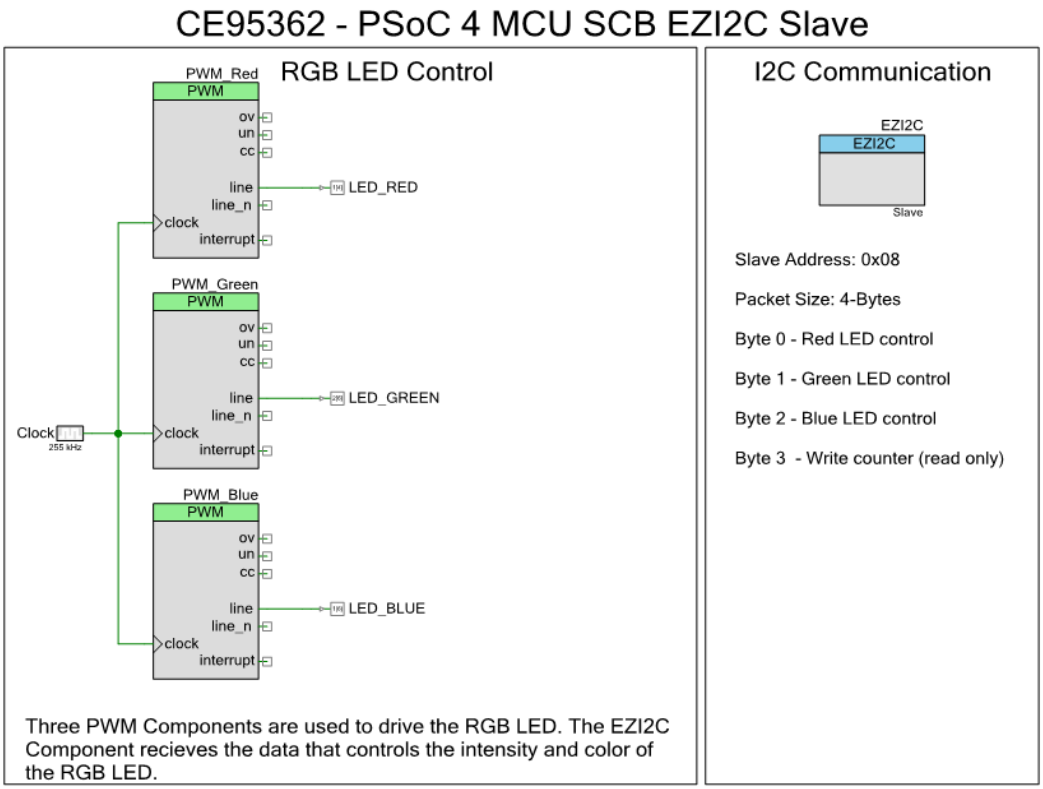
Figure 2. Commands Execution Results




## Design and Implementation

The Top Design Schematic of the project is shown in Figure 3.

Figure 3. PSoC Creator Project Schematic




EZI2C is an implementation of the standard I²C communication protocol in which a single buffer is shared between the Slave device and the Master device. When the buffer is initialized, a writable boundary index is specified. All indices in the buffer that are within the boundary index are writable. All data at or beyond the read/write boundary index is read-only. For example, creating a buffer with a size of 5 bytes and a writeable boundary at byte 3 would create the buffer shown below.

| Byte 0 | Byte 1 | Byte 2 | Byte 3 | Byte 4 |
|---|---|---|---|---|
| Read and Write | Read and Write | Read and Write | Read Only | Read Only |

In this example, the EZI2C Component is configured with a 4-byte buffer (memory), which can be accessed by the I²C Master. The first three bytes are writeable and hold the red, green, and blue LED intensity in that order. The fourth byte, which is read-only, holds the number of write operations performed after device reset.

EZI2C allows an I²C Master to either access an individual byte from the Slave memory (by specifying the memory address in the Write command) or all memory bytes at once. For the testing procedure, see Operation.

To control the color and intensity of an RGB LED, three PWMs with period value of 255 (~1 kHz) are used. Duty cycles of the PWMs are controlled in the firmware and specified by the I²C Master. Changing the duty cycle of the PWM signal results in a change in the LED intensity. By changing the intensity of individual LEDs, various colors can be produced on the RGB LED using color mixing. The range of intensity in this example is 0x01 to 0xFF; 0x00 turns the LED OFF. Note that the LEDs on the kit are open-drain devices that are driven low. To compensate for this, the inverting output of each TCPWM Component is used to drive the LEDs.

## Components and Settings

Table 3 lists the PSoC Creator Components used in this example, how they are used in the design, and the non-default settings required so they function as intended.

Table 3. PSoC Creator Components

| Component | Instance Name | Purpose | Non-default Settings |
|---|---|---|---|
| EZI2C (SCB mode) | EZI2C | Provides I²C register-based communication with the Master device | Default settings only |
| Digital Output Pin | LED_RED | Drives the PWM signal to the LED | Default settings only |
| | LED_GREEN | | Default settings only |
| | LED_BLUE | | Default settings only |
| Clock | Clock | Drives the PWM at 24 MHz | **Frequency:** 24 MHz |
| PWM (TCPWM mode) | PWM_Red | Generate square wave and bring out the signal to GPIO | **Period:** 255<br>**Compare:** 0 |
| | PWM_Green | | **Period:** 255<br>**Compare:** 0 |
| | PWM_Blue | | **Period:** 255<br>**Compare:** 0 |

For information on the hardware resources used by the Component, see the Component datasheet.

## Reusing This Example

This example is designed for the kits shown in Table 1. To port the design to a different PSoC 4 device and/or kit, change the target device using **Device Selector** and update the pin assignments in the **Design Wide Resources Pins** settings as needed.

## Related Documents

| | |
|---|---|
| **Application Notes** | |
| AN79953 – Getting Started with PSoC 4 | Introduces the PSoC 4 architecture and development tools. |
| **PSoC Creator Component Datasheets** | |
| Pins | Supports connection of hardware resources to physical pins |
| Serial Communication Block (SCB) | Supports the hardware SCB block |
| Timer/Counter Pulse Width Modulator (TCPWM) | Supports generation of Pulse Width Modulation signals |
| **Device Documentation** | |
| PSoC 4 Datasheets | PSoC 4 Technical Reference Manuals |
| **Development Kit (DVK) Documentation** | |
| PSoC 4 Kits | |

## Document History

Document Title: CE195362 – PSoC 4 EZI2C Slave with Serial Communication Block (SCB)

Document Number: 001-95362

| Revision | ECN | Orig. of Change | Submission Date | Description of Change |
|---|---|---|---|---|
| ** | 6000861 | MYKZTMP1 | 12/28/2017 | New code example |
| *A | 6095934 | BFMC | 03/14/2018 | Updated to closely match PSoC 6 EZI2C example<br>Updated template |
| *B | 6162776 | BFMC | 05/3/2018 | Updated Overview section to contain EZI2C Buffer initialization<br>Added Bridge Control Panel Help instructions<br>Added separate project for PSoC 4S-Series and PSoC Analog Coprocessor |
| *C | 6223938 | BFMC | 07/5/2018 | Updated pin assignments for CY8CKIT-042 |

## Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

## Products

| | |
|---|---|
| Arm® Cortex® Microcontrollers | cypress.com/arm |
| Automotive | cypress.com/automotive |
| Clocks & Buffers | cypress.com/clocks |
| Interface | cypress.com/interface |
| Internet of Things | cypress.com/iot |
| Memory | cypress.com/memory |
| Microcontrollers | cypress.com/mcu |
| PSoC | cypress.com/psoc |
| Power Management ICs | cypress.com/pmic |
| Touch Sensing | cypress.com/touch |
| USB Controllers | cypress.com/usb |
| Wireless Connectivity | cypress.com/wireless |

## PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

## Cypress Developer Community

Community Forums | Projects | Videos | Blogs | Training | Components

## Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2017-2018. 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. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. 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.