

SCB_EzI2cCommSlave Example Project

1.0

Features

- Communication between I²C master and EZ I²C slave
- Simple packet protocol with command and status byte
- EZ I²C slave sets to read only buffer area for packet with status

General Description

This example project demonstrates the basic operation of the EZ I²C slave (SCB mode) component. The EZ I²C slave accepts the packet with a command from the I²C master to control the RGB LED color. The EZ I²C slave updates its buffer with a status packet in response to the accepted command.

Development Kit Configuration

This example project is designed to be executed on CY8CKIT-042 from Cypress Semiconductor. A full description of the kit, along with more example programs and ordering information, can be found at <http://www.cypress.com/go/cy8ckit-042>.

The project requires configuration settings changes in order to run on CY8CKIT-040 from Cypress Semiconductor. A full description of the kit, along with more example programs and ordering information, can be found at <http://www.cypress.com/go/cy8ckit-040>.

In order to switch from CY8CKIT-042 to the CY8CKIT-040 following steps should be performed:

1. Change the project's device from CY8C4245AXI-483 to CY8C4014LQI-422 with a Device Selector called from the project's context menu.
2. Change the clock configuration. In the Workspace Explorer window, double-click the project's design-wide resource file and click on the **Edit Clocks...** icons on the **Clocks** tab. Set IMO frequency to **32 MHz**.
3. Change the assignment of the pin components to physical pins. In the Workspace Explorer window, double-click the project's design-wide resource file and assign the pins for EZ I²C slave and RGB LED accordingly to Table 1.

Table 1. Pin assignment of the SCB_EzI2cCommSlave project

Pin Name	Development Kit	
	CY8CKIT-042	CY8CKIT-040
\EZI2C:scl\	P3[0]	P1[2]
\EZI2C:sda\	P3[1]	P1[3]
LED_BLUE	P0[3]	P0[2]
LED_GREEN	P0[2]	P1[1]
LED_RED	P1[6]	P3[2]

Project Configuration

The example project consists of the EZ I²C slave (SCB mode) and pin components. The design schematic is shown in Figure 1. The blue annotation components are used to represent RGB LED instated on the kit. The three pin components are used to control the LED color. The kit provides connection of the EZ I²C slave (PSoC 4) and I²C master (PSoC 5LP) as well as a pull-up resistor required for the I²C bus operation. The Bridge Control Panel software is provided to control the I²C master.

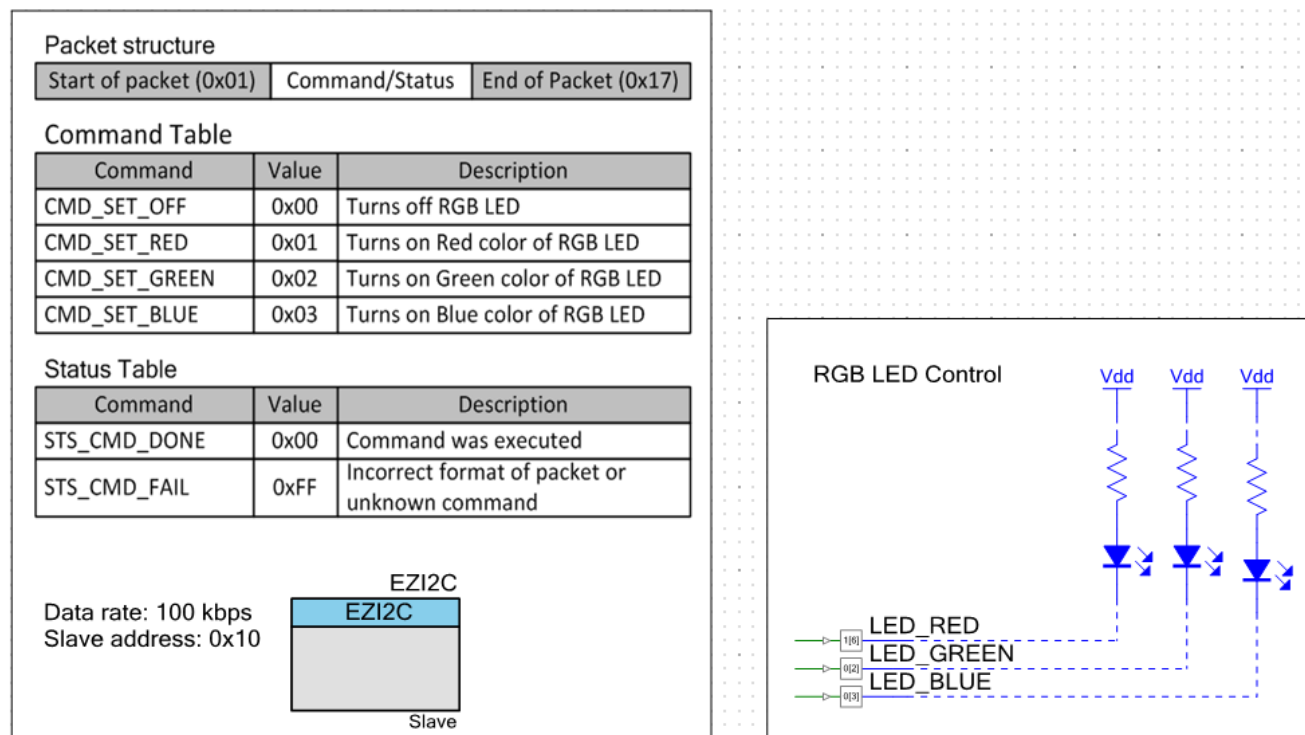


Figure 1. Example project design schematic

The EZ I²C slave is configured to operate with the data rate of 100 kbps and responds to address 0x10 (7-bits). The base address is set to 8 bits which allows the buffer addressing up to 256 bytes. The EZ I²C slave applies clock stretching on the bus in the case of late interrupt handling.

The component configuration window is shown below.

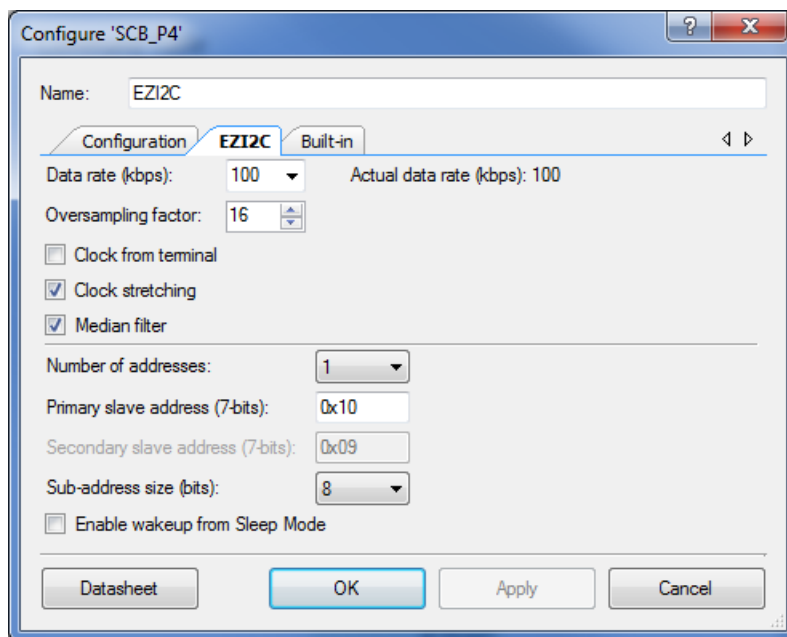


Figure 2. EZ I²C slave (SCB mode) component configuration

Project Description

In the main firmware routine, the EZ I²C slave buffer is configured when a component is started. The EZ I²C exposes a single buffer to the master. The buffer is divided into write/read areas to write a packet with a command and to read only the area to read to packet with a status. The RGB LED is turned off. The write/read area starts from base address 0 and the read only area from 3. Interrupts are enabled to the CPU core as required by the EZ I²C slave component for operation.

The EZ I²C slave waits for communication from the I²C master. The main loop polls I2C_EzI2CGetActivity () API continuously for a write completion event. When a write completion event is reported, the write/read area in the buffer content is checked against the valid packet. The packet structure and table with commands are shown below. The basic checks of the packet structure are done: the start and end of the packet byte. If a packet is considered as valid, the command is retrieved and passed to be executed. The result of a command execution is change of the LED color. Initially the LED is turned off. The status is updated with a successful command execution or failure when the command is unknown or the packet considered as invalid. The table with return statuses is shown below. The packet with a status is exposed to the master in the read only area of the slave buffer.

Packet structure

Start of packet (0x01)	Command/Status	End of Packet (0x17)
------------------------	----------------	----------------------

Table 2. Command constants

Command	Value	Description
CMD_SET_OFF	0	Turns off RGB LED
CMD_SET_RED	1	Turns on Red color of RGB LED
CMD_SET_GREEN	2	Turns on Green color of RGB LED
CMD_SET_BLUE	3	Turns on Blue color of RGB LED

Table 3. Status constants

Status	Value	Description
STS_CMD_DONE	0x00	Command was executed
STS_CMD_FAIL	0xFF	Incorrect format of packet or unknown command

The packets with a command and status are converted into the following I²C master transfers. The base address is followed by the EZ I²C slave address to access the appropriate buffer area. The packet with a command has a write direction set in the address byte and the base address is set to 0. The following bytes are written into the buffer start from the base address. To read a packet with a status, the base address has to be changed first. The master sends an address byte with a write direction and sets the base address to 0x03 but without any following data. Then the direction is changed to read in the address byte and the packet with a status is read.

Packet with command

S	ADDR = 0x10	W	A	Base Addr = 0x00	A	SOP = 0x01	A	Command	A	EOP = 0x17	A	P
---	-------------	---	---	------------------	---	------------	---	---------	---	------------	---	---

Packet with status

S	ADDR = 0x10	W	A	Base Addr = 0x03	A	Sr	ADDR = 0x10	R	A	SOP = 0x01	A	Status	A	EOP = 0x17	\bar{A}	P
---	-------------	---	---	------------------	---	----	-------------	---	---	------------	---	--------	---	------------	-----------	---


☐ - Master drives the bus ☒ - Slave drives the bus

Expected Results

Build and program the SCB_EzI2cCommSlave example project.

Run the Bridge Control Panel software which is shipped with the PSoC Creator. It is used to control the I²C master implemented on the PSoC 5LP available on the kit. Follow the steps above to set up communication between the master and slave:

1. Select the KitProg device, which was programmed with SCB_EzI2cCommSlave example project, from the list of the Connected Ports.
2. Make sure that the selected Protocol is I²C.
3. Go to Tools->Protocol Configuration and select I2C Speed 100kHz.

4. Press the List button  to make sure that the EZ I²C slave device with address 0x10 (7-bits) is available for communication¹.

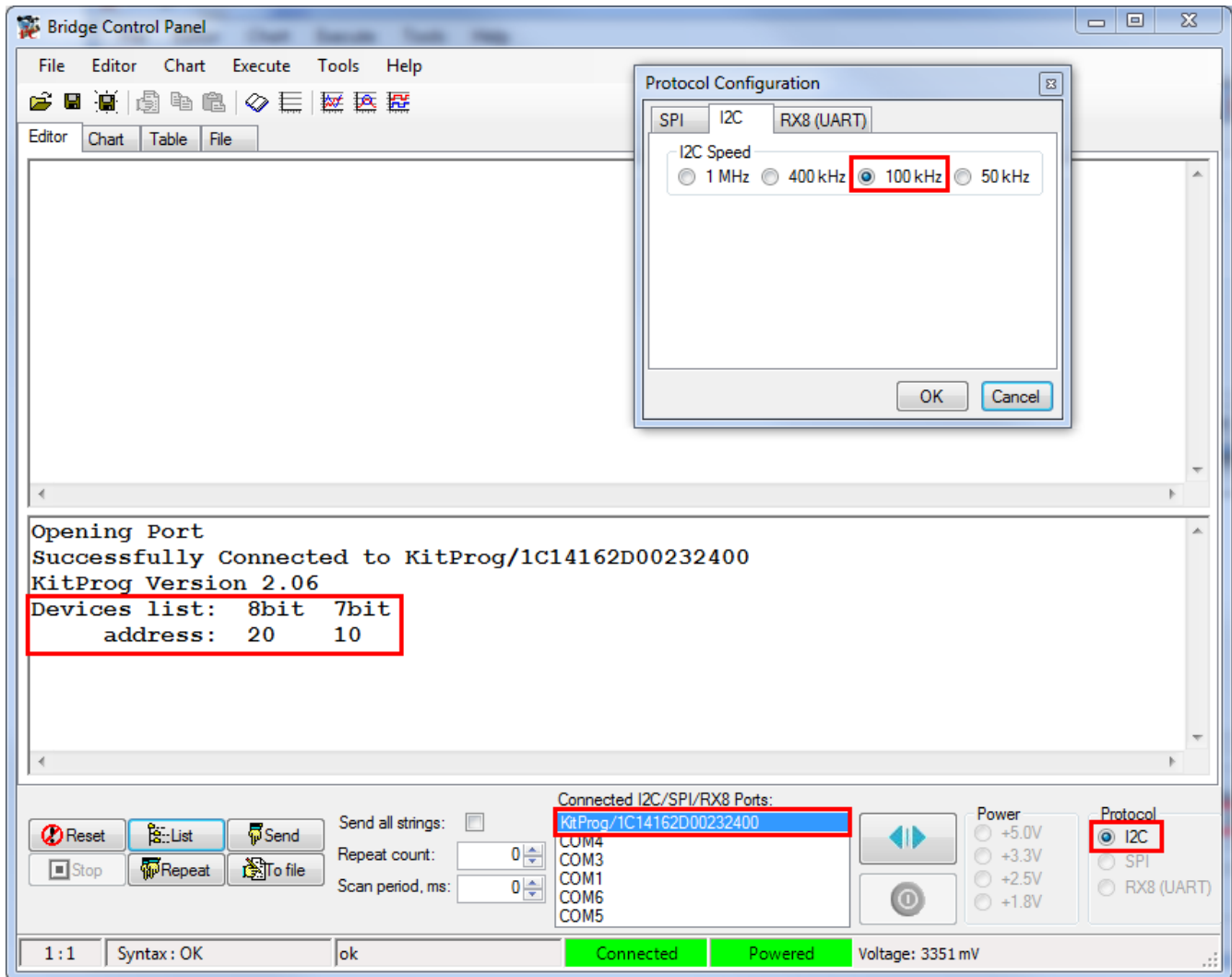



Figure 3. Bridge Control Panel I²C master setup

5. To load master commands for communication with the EZ I²C slave use Open icon  . Navigate to BCP_Master_EzI2cCmd.iic file which is attached to the workspace and open it. The commands should appear in the Edit window.
6. There are two options of a master transfer execution.

¹ The device with 7-bits address 0x50 appears on I²C when list command is executed for CY8CKIT-040. This I²C device is Serial F-RAM.

- A single command execution: set the cursor to the line with a command into the Edit window and press Enter. The RGB LED should change its color accordingly to the executed command.
- Repeat the command execution: select a number of commands and press the Repeat button. The RGB LED should change its color accordingly to the executed commands.

The delay between commands is added to be able to distinguish a LED color change.

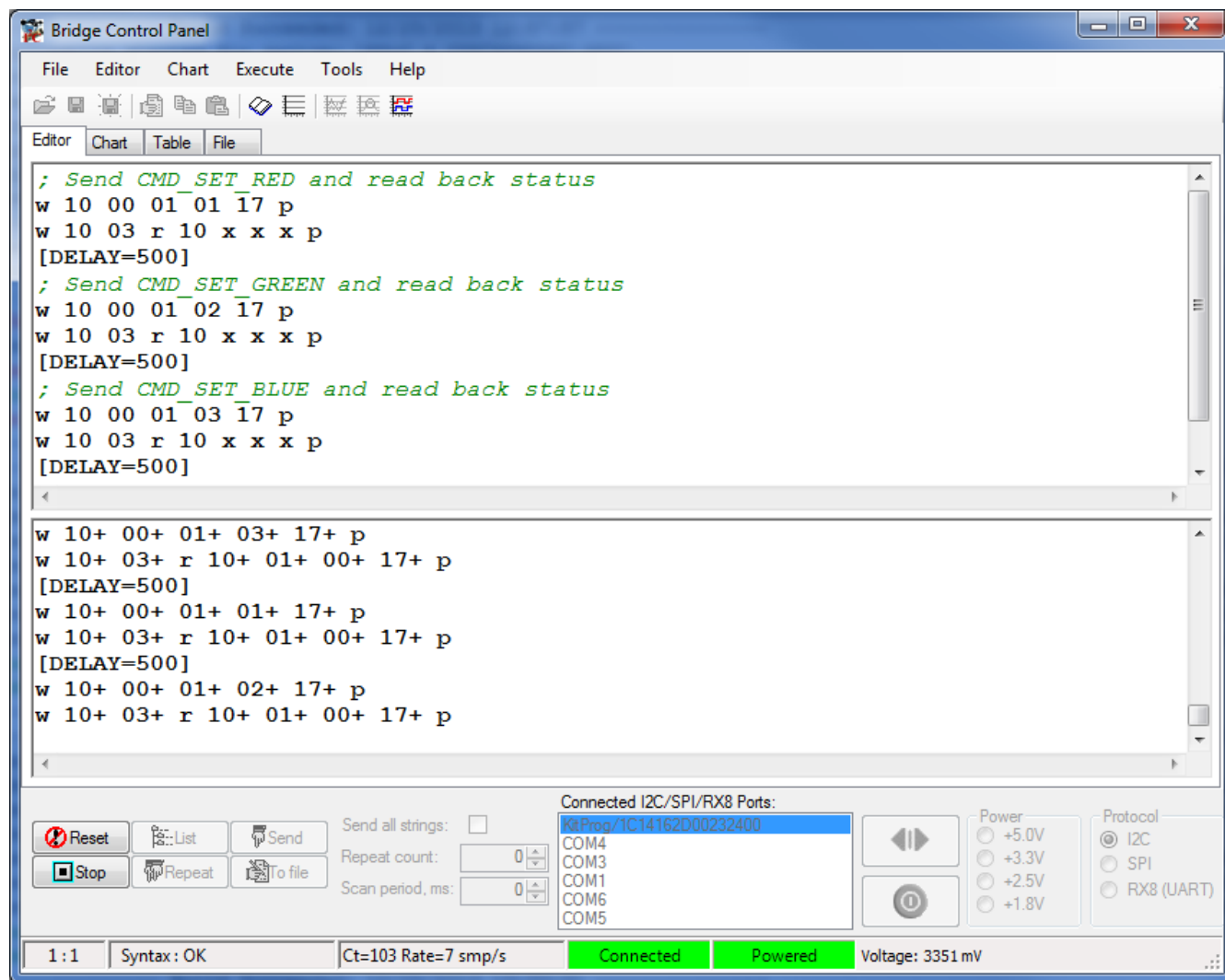


Figure 4. Repeat command execution result

© Cypress Semiconductor Corporation, 2014. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.