SCB_SpiCommSlave Example Project
1.0

Features

- Communication between SPI master (UDB) and SPI slave (SCB)
- Simple packet protocol with command and status byte to update RGB LED color

General Description

This example project demonstrates the basic operation of the SPI slave (SCB mode) component. The UDB-based SPI master initiates transfers to a SCB SPI slave. The SPI slave accepts a 3-bytes packet with a command from the master to control the RGB LED color. After the command execution, the slave updates its buffer with a status packet in response to the accepted command.

Development Kit Configuration

This example project is designed to run on the CY8CKIT-042 kit from Cypress Semiconductor. A 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 to run on other kits from Cypress Semiconductor. Table 1 is the list of the supported kits. To switch from CY8CKIT-042 to any other kit, change the project's device with the help of Device Selector called from the project's context menu.

Table 1. Development Kits vs Parts

Development Kit	Device
CY8CKIT-042	CY8C4245AXI-483
CY8CKIT-042-BLE	CY8C4247LQI-BL483
CY8CKIT-044	CY8C4247AZI-M485
CY8CKIT-046	CY8C4248BZI-L489

The pin assignments for the supported kits are in Table 2.

Table 2. Pin Assignment

Pin Name	Development Kit			
	CY8CKIT-042	CY8CKIT-042 BLE	CY8CKIT-044	CY8CKIT-046
\SPIS:miso_s\	P3[1]	P0[1]	P6[1]	P6[1]
\SPIS:mosi_s\	P3[0]	P0[0]	P6[0]	P6[0]
\SPIS:sclk_s\	P0[6]	P0[3]	P6[2]	P6[2]
\SPIS:ss0_s\	P0[7]	P0[2]	P2[7]	P6[3]

Pin Name	Development Kit			
	CY8CKIT-042	CY8CKIT-042 BLE	CY8CKIT-044	CY8CKIT-046
miso_m	P2[1]	P2[1]	P2[1]	P2[1]
mosi_m	P2[2]	P2[2]	P2[2]	P2[2]
sclk_m	P2[0]	P2[0]	P2[0]	P2[0]
ss_m	P2[3]	P2[3]	P2[3]	P2[3]
LED_BLUE	P0[3]	P3[7]	P6[5]	P5[4]
LED_GREEN	P0[2]	P3[6]	P2[6]	P5[3]
LED_RED	P1[6]	P2[6]	P0[6]	P5[2]

The appropriate SPI master and slave pins must be connected using external wires as described in the following table:

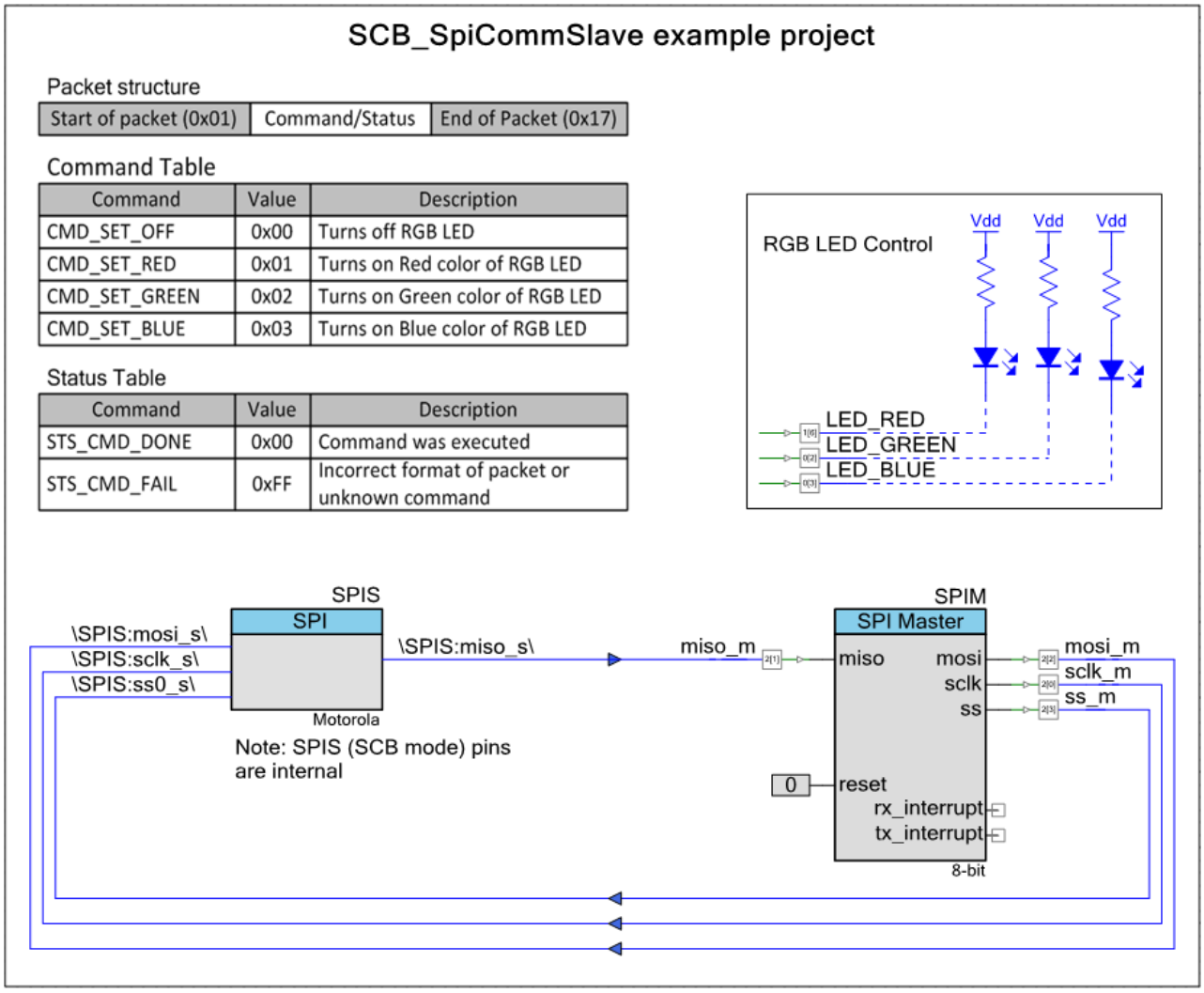
Table 3. Required External Connections

Master pins	Slave pins	Connection
miso_m	\SPIS:miso_s\	Connect MISO pins of master and slave
mosi_m	\SPIS:mosi_s\	Connect MOSI pins of master and slave
sclk_m	\SPIS:sclk_s\	Connect SCLK pins of master and slave
ss_m	\SPIS:ss0_s\	Connect SS pins of master and slave

Project Configuration

The example project consists of the SPI master (UDB), SPI slave (SCB mode) and pin components. **Figure 1** is the design schematic. The SPI master and slave connections are in the design schematic. The blue annotation components represent the RGB LED installed on the kit. Three pin components control the LED color, using the fixed connections already provided on the kit.

Figure 1. Example Project Design Schematic



The SPI slave operates with the bit rate of 1000 kbps. The sub-mode is Motorola, CPHA = 0, CPOL = 0, and Data width = 8 bits with Bit Order = MSB first. The packet size to transfer is less than 8 data elements, therefore only the hardware FIFO is used (the FIFO depth is 8 data elements[1]). The component configuration window is shown below.

[1] All devices except PSoC 4100/PSoC 4200 provide the ability to double the FIFO depth to be 16 data elements when the data frame width is 4-8 bits.

Figure 2. SPI Slave (SCB mode) Component Configuration Basic Tab

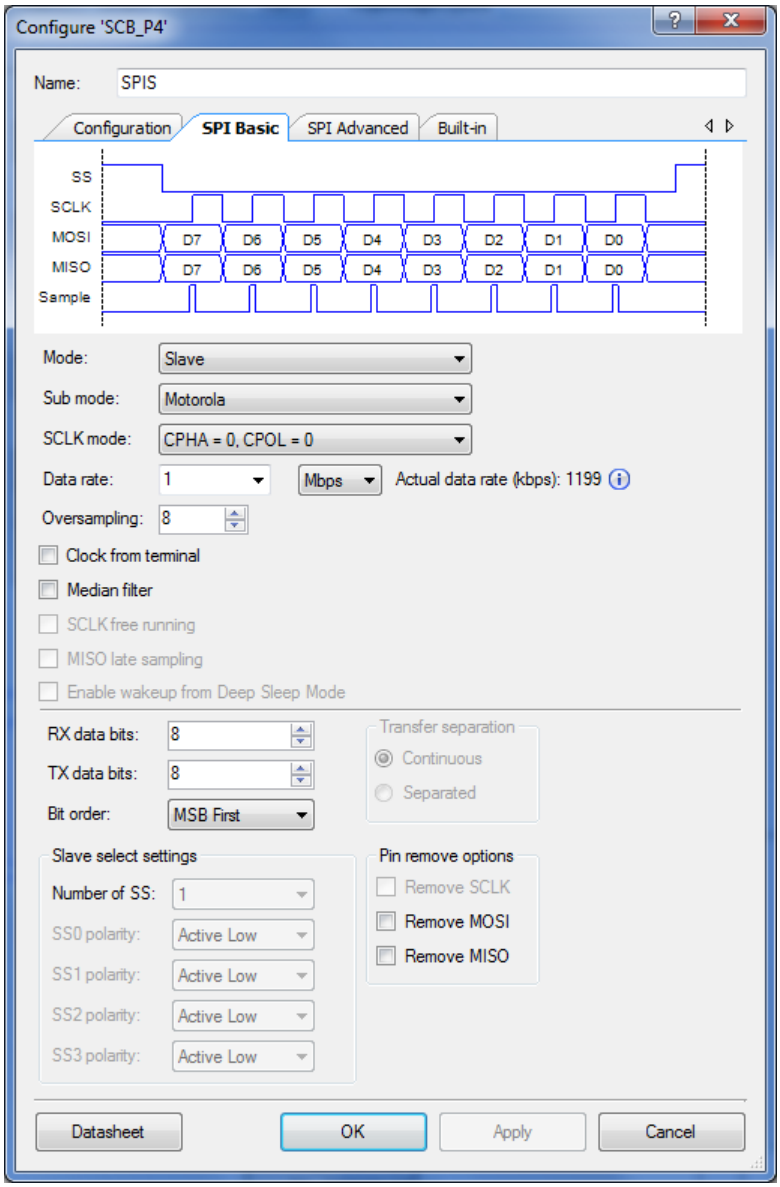
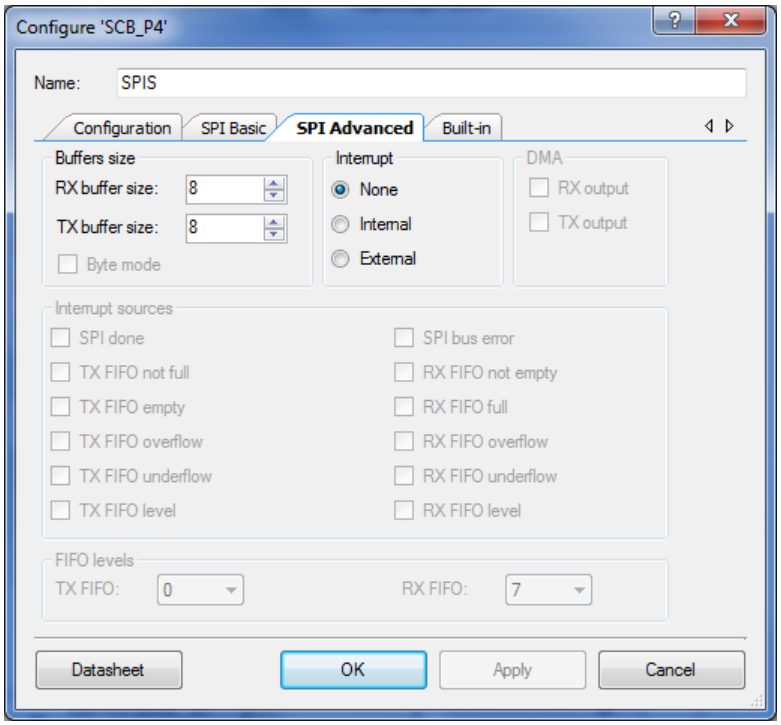


Figure 3. SPI Slave (SCB mode) Component Configuration Advanced Tab



If you need to transfer more data elements than the FIFO depth[1], increase the buffer size to the required amount in Advanced Tab. This enables the hardware FIFO and a circular software buffer. The circular software buffer requires enabling global interrupts as the component interrupt transfers data between the hardware FIFO and allocated software buffer.

To see the SPI master configuration, open the SPIM component configuration dialog.

Project Description

The SPI master and slave are combined into a single project and events from them are handled sequentially. The master sends a packet with a RGB color command; the slave receives it and updates LED accordingly. After command execution, the slave updates the status byte in the response packet. The master transfers the response packet to retrieve the status from the slave (the slave cannot initiate the transfer to send the status to the master). If needed, the project can be divided into separate master and slave projects with minimum modifications.

In the main firmware routine, the SPI master and slave configuration is executed by calling the start APIs. LEDs are turned off. Interrupts are not used in this example; therefore enabling global interrupts is not required.

In the main loop, the SPI master communicates with the slave every 500 milliseconds to change the LED color. The SPI master initializes a packet with a command and sets up a transfer by loading a packet into the TX buffer. The initial command is *CMD_SET_RED*. The packet structure, commands, and responses are shown below.

Master Operation

The code polls for the status of the SPI_DONE interrupt source by calling *SPIM_ReadTxStatus().* This interrupt source triggers when a transfer is complete. After transfer completion, the data in the RX buffer is thrown away as it contains dummy data bytes transferred by the slave. When the slave executes the command, the master initiates a transfer to get a status response packet, by loading dummy data bytes into the TX buffer. After the transfer is complete, the response packet is read from the RX buffer. The basic verification of the packet structure is done by checking the start and end packet bytes. The status byte is checked afterwards. If all the checks are successful, the command is considered as executed and the master continues to execute the next command. The command sequence is the following: CMD_SET_RED, CMD_SET_GREEN, CMD_SET_BLUE, CMD_SET_OFF, CMD_SET_RED and so on.

Slave Operation

The SPI slave waits for a packet with a command from the master. The code polls the RX buffer sizes calling *SPIS_SpiUartGetRxBufferSize()*, when the buffer size is equal to the packet size, the transfer is treated as completed. When a packet is received, the packet is verified by checking the start and end packet bytes. The command from a valid packet is passed to the *ExecuteCommand()* function which sets the LED color and returns a status. The status is updated in the response packet and the SPI slave loads it into the TX buffer to be read by the master. After the response has been read, the slave TX buffer is cleared before accepting the next command packet.

Packet structure

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

Table 3. Command Constants

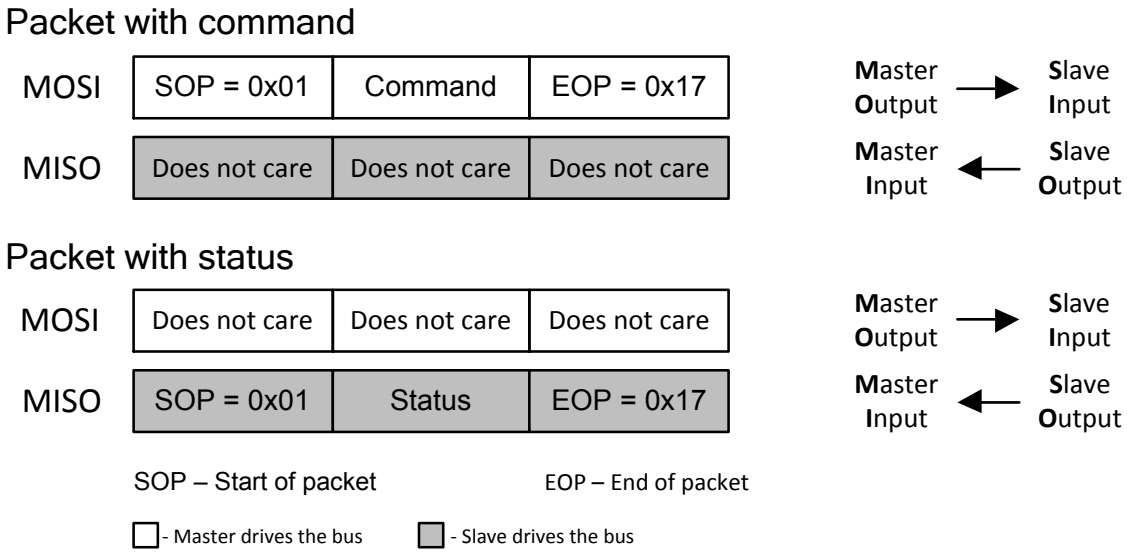
Command	Value	Description
CMD_SET_OFF	0	Turns off RGB LED
CMD_SET_RED	1	Turns on Red color of RGB LED
CMD_SET_GREEN	2	Turns on Green color of RGB LED
CMD_SET_BLUE	3	Turns on Blue color of RGB LED

Table 4. 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 SPI transfers. Because the SPI interface is full-duplex, the data elements are transmitted and received simultaneously. Data elements received by the master while it is transferring the command packet are discarded.

The same approach works for the slave. Data elements received by the slave while the master is collecting the status packet are discarded.



Expected Results

1) Connect the SPI master and slave as explained in the Development Kit Configuration section.
2) Build example project and program into the device.
3) Observe that LED changes its color in the following sequence: RED, GREEN, BLUE, OFF, RED and so on. An oscilloscope or SPI bus analyzer could be used to capture traffic on the SPI bus.