How to use serial communications block (SCB) in TRAVEO™ T2G family

About this document

Scope and purpose
AN225401 demonstrates how to configure and use a serial communications block (SCB) in TRAVEO™ T2G family MCU with three serial interface protocols: SPI, UART, and I2C.

Associated part family
TRAVEO™ T2G Family CYT2/CYT3/CYT4 series

Intended audience
This document is intended for anyone using TRAVEO™ T2G family.

Table of contents

1 Introduction
1.1 Features
2 General description
3 SPI setting procedure example
3.1 Master mode
3.1.1 Use case
3.1.2 Configuration and example
3.2 Slave mode
3.2.1 Use case
3.2.2 Configuration and example
4 UART setting procedure example
4.1 UART mode
4.1.1 Use case
4.1.2 Configuration and example
5 I2C setting procedure example
5.1 Master mode
5.1.1 Use case
5.1.2 Configuration and example
5.2 Slave mode
5.2.1 Use case
5.2.2 Configuration and example
6 Glossary
7 Related documents
# Table of contents

8  Other references ................................................................. 81  
Revision history ................................................................. 82  
Disclaimer ................................................................. 83
1 Introduction

Introduction

This application note describes how to use a serial communications block (SCB) in TRAVEO™ T2G CYT2/CYT3/CYT4 series MCUs. The SCB is used for serial communication with other devices; it supports three serial communication protocols: SPI, UART, and I2C.

This application note explains the functioning of SCB, initial configuration, and data communication operations with use cases. To understand the functionality described and terminology used in this application note, see the “Serial Communications Block (SCB)” chapter of the architecture technical reference manual (TRM).

1.1 Features

The SCB supports the following features:

- Standard SPI Master and Slave functionality with Motorola, Texas Instruments, and National Semiconductor protocols
- Standard UART functionality with SmartCard reader, Local Interconnect Network (LIN), and IrDA protocols
  - Standard LIN Slave functionality with LIN v1.3 and LIN v2.1/2.2 specification compliance. The SCB in TRAVEO™ T2G family has only standard LIN Slave functionality.
- Standard I2C Master and Slave functionality
- Only SCB[0] is DeepSleep-capable
- EZ mode for SPI and I2C Slaves allows for operation without CPU intervention
- CMD_RESP mode for SPI and I2C Slaves allows for operation without CPU intervention, and available only in DeepSleep-capable SCB
- Low-power (DeepSleep) mode of operation for SPI and I2C Slaves (using external clocking), only available on DeepSleep-capable SCB
- DeepSleep wakeup on I2C Slave address match or SPI Slave selection; only available on DeepSleep-capable SCB
- Trigger outputs for connection to DMA
- Multiple interrupt sources to indicate status of FIFOs and transfers
2 General description

The SCB supports three serial communication protocols: SPI, UART, and I2C. Only one of the protocols is supported by an SCB at any given time.

The SCB supports only the Slave functions of the LIN standard. Therefore, UART-LIN of the SCB cannot be used for the LIN Master. For details on the supported hardware and LIN Master tasks, see the description of the LIN block in the architecture technical TRM.

Figure 1 shows the block diagram of SCB.

**Figure 1**  Block diagram of SCB

The SCB consists of registers, FIFO, and a control block for each protocol function (SPI, UART, and I2C). Registers are used as software interfaces for SCB settings and generated interrupts by each event. The FIFO consists of SRAM (256-byte) and has three modes Tx/Rx FIFO (128x8-bit/ 64x16-bit/ 32x32-bit), EZ (256x8-bit), and command/response (256x8-bit). Each protocol function control block works as a transmitting and receiving controller. SPI (all SCBs) and I2C (SCB[0]) support externally clocked (EC) mode in Slave mode.
3 SPI setting procedure example

This section shows an example of the SPI using the sample driver library (SDL). The SCB supports SPI Master mode and SPI Slave mode with Motorola, Texas Instruments, and National Semiconductor protocols. See the architecture technical TRM for details of each protocol. The code snippets in this application note are part of SDL, and are based on CYT2B7 series. See Other references for the SDL.

The SDL has a configuration part and a driver part. The configuration part mainly configures the parameter values for the desired operation. The driver part configures each register based on the parameter values in the configuration part. You can configure the configuration part according to your system.

3.1 Master mode

This example configures the SCB in Motorola SPI Master mode and transmits two words (word: 16 bits) of data and receives two words of optional data from the SPI Slave.

3.1.1 Use case

The following is an example of sending data in the scheduler cycle (100 ms) and receiving data in the Rx interrupt.

- SCB mode = Motorola SPI Master mode
- SCB channel = 1
- PCLK (peripheral clock) = 4 MHz
- Bit rate = 1 Mbps (OVS: oversampled multiple. See the “Serial Communications Block (SCB)” chapter in the architecture technical TRM.)

[Bit rate setting]

The bit rate setting is valid only in Master mode. The formula of bit rate calculation is as follows: Bit rate [bps] = Input clock [Hz] / OVS OVS : SCB_CTRL.OVS + 1. In this case, bit rate is calculated as follows: Bit rate = Input clock [Hz] / OVS = PCLK(4MHz) / (3+1) = 1 [Mbps]. For more details, see the architecture technical TRM.

- Tx/Rx data length = 16 bits
- Tx/Rx FIFO = Used (16-bit FIFO data elements)
- Rx interrupt = Enable
  - System interrupt source : scb_0_interrupt_IRQn (IDX: 17)
  - Mapped to CPU interrupt : IRQ3
  - CPU interrupt priority : 3
- Used ports
  - SCLK : SCB0_CLK (P0.2)

MOSI data is driven on a falling edge of SCLK
MISO data is captured on a falling edge of SCLK after half a SPI SCLK period from a rising edge.

Figure 2 shows the example of connection between the SCB and another SPI device.

![Figure 2](Example of SPI (Master mode) communication)
In SPI mode, SCLK, MOSI, MISO, and SELECT signals are connected to another Slave device. In Master mode, SCLK and MOSI are output, and MISO is input. SELECT is used as an indication of valid data period for the Slave device. Up to four SELECT signals can be assigned. Figure 3 shows the setting procedure and operation example for Master mode.

**Figure 3  SPI Master mode operation**

(1) Configure the clock
(2) Enable global interrupt
(3) Set the parameters to interrupt structure
(4) Set the system interrupt handler
(5) Interrupt Enable
(6) Set the SPI port
(7) Initialize SCB for SPI communication
(8) Enable SPI
(9) Initialize the scheduler

Interrupt Handler (irqSCB)

(10) Check the RX FIFO Interrupt is Active
(11) Read the Received SPI Data
(12) Clear the RX FIFO Interrupt Flag

Interrupt Handler (SCB_TickHandler)

(13) Write and send the Received SPI Data

(14) END

In SPI mode, SCLK, MOSI, MISO, and SELECT signals are connected to another Slave device. In Master mode, SCLK and MOSI are output, and MISO is input. SELECT is used as an indication of valid data period for the Slave device. Up to four SELECT signals can be assigned. Figure 3 shows the setting procedure and operation example for Master mode.

(1) Configure the clock
(2) Enable global interrupt (CPU interrupt enable). For details, see the CPU interrupt handing sections in the architecture technical TRM.
(3) Set the interrupt structure. For details, see the CPU interrupt handing sections in the architecture technical TRM.
(4) Set the system interrupt handler. For details, see the CPU interrupt handing sections in the architecture technical TRM.
(5) Enable the interrupt.
3 SPI setting procedure example

(6) Set the SPI port for Master mode. SCLK, MOSI, and SELECT are output. MISO is input.
(7) Initialize the SCB for SPI communication.
(8) Enable SPI.
(9) Initialize and start the scheduler for SPI transmit data.
(10) When the Master receives optional data, an RX FIFO interrupt occurs.
(11) The software reads the received data from the Rx FIFO.
(12) Clear the RX FIFO interrupt.
(13) When a scheduler interrupt occurs, write the 16-bit data to be transmitted to the SCB_TX_FIFO_WR register. The SCB will start transmitting as soon two or more bytes are written to the FIFO. (SCB_TX_FIFO_WR = transmit_data.)

3.1.2 Configuration and example

Table 1 lists the parameters of the configuration part in the SDL for SPI Master mode.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCE_CLOCK_FRQ</td>
<td>Input divider clock frequency</td>
<td>800000000ul (80MHz)</td>
</tr>
<tr>
<td>SCB_SPI_BAUDRATE</td>
<td>SPI baud rate</td>
<td>125000ul</td>
</tr>
<tr>
<td>SCB_SPI_OVERSAMPLING</td>
<td>Oversampling for SPI</td>
<td>16ul</td>
</tr>
<tr>
<td>SCB_SPI_CLOCK_FREQ</td>
<td>Peripheral clock frequency</td>
<td>SCB_SPI_BAUDRATE<em>SCB_SPI_OVERSAMPLING (125000</em>16 = 2 MHz)</td>
</tr>
<tr>
<td>DIVIDER_NO_1</td>
<td>Divider number</td>
<td>1ul</td>
</tr>
<tr>
<td>CY_SPI_SCB_PCLK</td>
<td>Peripheral clock number</td>
<td>PCLK_SCB0_CLOCK</td>
</tr>
<tr>
<td>CY_SPI_SCB_TYPE</td>
<td>SCB channel number</td>
<td>SCB0</td>
</tr>
<tr>
<td>SCB_SPI_cfg.spiMode</td>
<td>Operation mode</td>
<td>CY_SPI_SCB_MASTER (0x1)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.subMode</td>
<td>SPI operation sub mode</td>
<td>CY_SPI_SCB_MOTOROLA (0x0)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.sclkMode</td>
<td>Clock polarity and phase</td>
<td>CY_SPI_SCB_CPHA0_CPOL0 (0x0)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.oversample</td>
<td>Oversampling factor</td>
<td>SCB_SPI_OVERSAMPLING (16ul)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.rxDataWidth</td>
<td>Rx data frame width</td>
<td>16ul</td>
</tr>
<tr>
<td>SCB_SPI_cfg.txDataWidth</td>
<td>Tx data frame width</td>
<td>16ul</td>
</tr>
<tr>
<td>SCB_SPI_cfg.enableMsbFirst</td>
<td>Least or most significant bit first</td>
<td>true (MSB)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.enableFreeRunSclk</td>
<td>SCLK generated mode</td>
<td>false</td>
</tr>
<tr>
<td>SCB_SPI_cfg.enableInputFilter</td>
<td>Median filter</td>
<td>false</td>
</tr>
<tr>
<td>SCB_SPI_cfg.enableMisoLateSample</td>
<td>SCLK edge on which MISO is captured</td>
<td>true (determined by CPOL and CPHA)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.enableTransferSeparation</td>
<td>Continuous SPI data transfers</td>
<td>true (disable)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.ssPolarity0</td>
<td>Polarity of SELECT 1</td>
<td>false (‘0’ active, used)</td>
</tr>
</tbody>
</table>

(table continues...)
### Table 1 (continued) SPI Master mode configuration parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCB_SPI_cfg.ssPolarity1</td>
<td>Polarity of SELECT 2</td>
<td>false ('0' active, unused)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.ssPolarity2</td>
<td>Polarity of SELECT 3</td>
<td>false ('0' active, unused)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.ssPolarity3</td>
<td>Polarity of SELECT 4</td>
<td>false ('0' active, unused)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.enableWakeFromSleep</td>
<td>Slave selection detection logic</td>
<td>false</td>
</tr>
<tr>
<td>SCB_SPI_cfg.rxFifoTriggerLevel</td>
<td>Trigger level of Rx FIFO</td>
<td>1ul</td>
</tr>
<tr>
<td>SCB_SPI_cfg.rxFifoIntEnableMask</td>
<td>Interrupt of Rx FIFO</td>
<td>1ul</td>
</tr>
<tr>
<td>SCB_SPI_cfg.txFifoTriggerLevel</td>
<td>Trigger level of Tx FIFO</td>
<td>0ul</td>
</tr>
<tr>
<td>SCB_SPI_cfg.txFifoIntEnableMask</td>
<td>Interrupt of Tx FIFO</td>
<td>0ul</td>
</tr>
<tr>
<td>SCB_SPI_cfg.enableSpiDoneInterrupt</td>
<td>SPI master transfer done event</td>
<td>false</td>
</tr>
<tr>
<td>SCB_SPI_cfg.enableSpiBusErrorInterrupt</td>
<td>SPI slave deselected at an unexpected time in the SPI transfer</td>
<td>false</td>
</tr>
<tr>
<td>CY_SPI_SCB_MISO_PORT</td>
<td>I/O port number</td>
<td>GPIO_PRT0</td>
</tr>
<tr>
<td>CY_SPI_SCB_MISO_PIN</td>
<td>I/O pin number</td>
<td>0ul</td>
</tr>
<tr>
<td>CY_SPI_SCB_MISO_MUX</td>
<td>Peripheral connection</td>
<td>P0_0_SCB0_SPI_MISO (30ul)</td>
</tr>
<tr>
<td>CY_SPI_SCB_MOSI_PORT</td>
<td>I/O port number</td>
<td>GPIO_PRT0</td>
</tr>
<tr>
<td>CY_SPI_SCB_MOSI_PIN</td>
<td>I/O pin number</td>
<td>1ul</td>
</tr>
<tr>
<td>CY_SPI_SCB_MOSI_MUX</td>
<td>Peripheral connection</td>
<td>P0_1_SCB0_SPI_MOSI</td>
</tr>
<tr>
<td>CY_SPI_SCB_CLK_PORT</td>
<td>I/O port number</td>
<td>GPIO_PRT0</td>
</tr>
<tr>
<td>CY_SPI_SCB_CLK_PIN</td>
<td>I/O pin number</td>
<td>2ul</td>
</tr>
<tr>
<td>CY_SPI_SCB_CLK_MUX</td>
<td>Peripheral connection</td>
<td>P0_2_SCB0_SPI_CLK</td>
</tr>
<tr>
<td>CY_SPI_SCB_SEL0_PORT</td>
<td>I/O port number</td>
<td>GPIO_PRT0</td>
</tr>
<tr>
<td>CY_SPI_SCB_SEL0_PIN</td>
<td>I/O pin number</td>
<td>3ul</td>
</tr>
<tr>
<td>CY_SPI_SCB_SEL0_MUX</td>
<td>Peripheral connection</td>
<td>P0_3_SCB0_SPI_SELECT0</td>
</tr>
<tr>
<td>SCB_MISO_DRIVE_MODE</td>
<td>DRIVE_MODE for MISO</td>
<td>CY_GPIO_DM_HIGHZ (0x08)</td>
</tr>
<tr>
<td>SCB_MOSI_DRIVE_MODE</td>
<td>DRIVE_MODE for MOSI</td>
<td>CY_GPIO_DM_STRONG_IN_OFF (0x06)</td>
</tr>
<tr>
<td>SCB_CLK_DRIVE_MODE</td>
<td>DRIVE_MODE for CLK</td>
<td>CY_GPIO_DM_STRONG_IN_OFF (0x06)</td>
</tr>
<tr>
<td>SCB_SEL0_DRIVE_MODE</td>
<td>DRIVE_MODE for SEL0</td>
<td>CY_GPIO_DM_STRONG_IN_OFF (0x06)</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.outVal</td>
<td>Pin output state</td>
<td>0ul</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.driveMode</td>
<td>GPIO drive mode</td>
<td>0ul</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.hsiom</td>
<td>Connection for I/O pin route</td>
<td>HSIOM_SEL_GPIO (0x0)</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.intEdge</td>
<td>Edge which will trigger an IRQ</td>
<td>0ul</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.intMask</td>
<td>Masks edge interrupt</td>
<td>0ul</td>
</tr>
</tbody>
</table>

*(table continues...)*
Table 1  (continued) SPI Master mode configuration parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI_port_pin_cfg.vtrip</td>
<td>Input buffer mode</td>
<td>0ul</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.slewRate</td>
<td>Slew rate</td>
<td>0ul</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.driveSel</td>
<td>GPIO drive strength</td>
<td>0ul</td>
</tr>
<tr>
<td>CY_SPI_SCB_IRQN</td>
<td>System interrupt index number</td>
<td>scb_0_interruptIRQn (IDX: 17)</td>
</tr>
<tr>
<td>irq_cfg.sysIntSrc</td>
<td>System interrupt index number</td>
<td>CY_SPI_SCB_IRQN</td>
</tr>
<tr>
<td>irq_cfg.intIdx</td>
<td>CPU interrupt number</td>
<td>CPUIntIdx3_IRQn</td>
</tr>
<tr>
<td>.isEnabled</td>
<td>CPU interrupt enable</td>
<td>true (Enable)</td>
</tr>
</tbody>
</table>

Table 2 lists the functions of the driver part in the SDL.

Table 2  List of functions

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cy_SysClk_PeriphAssignDivider</td>
<td>Assigns a programmable divider to a selected IP block</td>
<td>ipBlock: CY_SPI_SCB_PCLK</td>
</tr>
<tr>
<td>(en_clk_dst_t ipBlock, cy_en_divider_types_t dividerType, uint32_t dividerNum)</td>
<td>dividerType: CY_SYSCLK_DIV_24_5_BIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dividerNum: DIVIDER_NO_1</td>
<td></td>
</tr>
<tr>
<td>Cy_SysClk_PeriphSetFracDivider</td>
<td>Sets one of the programmable clock dividers</td>
<td>dividerType: CY_SYSCLK_DIV_24_5_BIT</td>
</tr>
<tr>
<td>(cy_en_divider_types_t dividerType, uint32_t dividerNum, uint32_t dividerIntValue, uint32_t dividerFracValue)</td>
<td>dividerNum: DIVIDER_NO_1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dividerIntValue: ((divSetting &gt;&gt; 5ul) &amp; 0x00000FFFul) - 1ul</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dividerFracValue: divSetting &amp; 0x0000001Ful</td>
<td></td>
</tr>
<tr>
<td>Cy_SysClk_PeriphEnableDivider</td>
<td>Enables the selected divider</td>
<td>dividerType: CY_SYSCLK_DIV_24_5_BIT</td>
</tr>
<tr>
<td>(cy_en_divider_types_t dividerType, uint32_t dividerNum)</td>
<td>dividerNum: 1ul</td>
<td></td>
</tr>
</tbody>
</table>

Table 3 lists the SPI functions of the driver part in the SDL.

Table 3  SPI functions

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cy_SCB_SPI_DeInit</td>
<td>De-initializes the SCB block</td>
<td>*base: CY_SPI_SCB_TYPE</td>
</tr>
<tr>
<td>(volatile stc_SCB_t *base)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cy_SCB_SPI_Init</td>
<td>Initializes the SCB for SPI operation</td>
<td>*base: CY_SPI_SCB_TYPE</td>
</tr>
<tr>
<td>(volatile stc_SCB_t *base, cy_stc_scb_spi_config_t const *config, cy_stc_scb_spi_context_t *context)</td>
<td>*config: SCB_SPI_cfg</td>
<td></td>
</tr>
<tr>
<td></td>
<td>context: NULL</td>
<td></td>
</tr>
<tr>
<td>Cy_SCB_SPI_SetActiveSlaveSelect</td>
<td>Selects an active Slave Select line from one of four available</td>
<td>*base: CY_SPI_SCB_TYPE</td>
</tr>
<tr>
<td>(volatile stc_SCB_t *base, uint32_t slaveSelect)</td>
<td>slaveSelect: 0ul</td>
<td></td>
</tr>
<tr>
<td>Cy_SCB_SPI_Enable</td>
<td>Enables the SCB block for the SPI operation</td>
<td>*base: CY_SPI_SCB_TYPE</td>
</tr>
<tr>
<td>(volatile stc_SCB_t *base)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 3 (continued) SPI functions

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Cy_SCB_SPI_GetRxFifoStatus</code></td>
<td>Returns the current status of the Rx FIFO</td>
<td>*base: CY_SPI_SCB_TYPE</td>
</tr>
<tr>
<td>(<code>volatile stc_SCB_t const *base</code>)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Cy_SCB_SPI_ReadArray</code></td>
<td>Reads an array of data out of the SPI Rx FIFO</td>
<td>*base: CY_SPI_SCB_TYPE</td>
</tr>
<tr>
<td>(<code>volatile stc_SCB_t const *base, void *rxBuf,</code></td>
<td></td>
<td><em>rxBuf: (void</em>)readData</td>
</tr>
<tr>
<td><code>uint32_t size)</code>)</td>
<td></td>
<td>size: 2ul</td>
</tr>
<tr>
<td><code>Cy_SCB_SPI_ClearRxFifoStatus</code></td>
<td>Clears the selected status of the Rx FIFO</td>
<td>*base: CY_SPI_SCB_TYPE</td>
</tr>
<tr>
<td>(<code>volatile stc_SCB_t *base,</code></td>
<td></td>
<td>clearMask:</td>
</tr>
<tr>
<td><code>uint32_t clearMask)</code>)</td>
<td></td>
<td>CY_SCB_SPI_RX_TRIGGER</td>
</tr>
<tr>
<td><code>Cy_SCB_SPI_WriteArray</code></td>
<td>Places an array of data in the SPI Tx FIFO</td>
<td>*base: CY_SPI_SCB_TYPE</td>
</tr>
<tr>
<td>(<code>volatile stc_SCB_t *base,</code></td>
<td></td>
<td><em>txBuf: (void</em>)readData</td>
</tr>
<tr>
<td><code>void *txBuf,</code></td>
<td></td>
<td>size: 2ul</td>
</tr>
<tr>
<td><code>uint32_t size)</code>)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Code Listing 1* demonstrates an example to configure SPI Master mode in the configuration part.
How to use serial communications block (SCB) in TRAVEO™ T2G family

3 SPI setting procedure example

Code Listing 1 Example to configure SPI Master mode in configuration part

```c
/* Device Specific Settings */
#define CY_SPI_SCB_TYPE SCB0  /* Define the SCB channel */

#define CY_SPI_SCB_MISO_PORT GPIO_PRT0  /* Define the port settings */
#define CY_SPI_SCB_MISO_PIN 0ul  /* Define the port settings */
#define CY_SPI_SCB_MISO_MUX P0_0_SCB0_SPI_MISO  /* Define the port settings */
#define CY_SPI_SCB_MOSI_PORT GPIO_PRT0  /* Define the port settings */
#define CY_SPI_SCB_MOSI_PIN 1ul  /* Define the port settings */
#define CY_SPI_SCB_MOSI_MUX P0_1_SCB0_SPI_MOSI  /* Define the port settings */
#define CY_SPI_SCB_CLK_PORT GPIO_PRT0  /* Define the port settings */
#define CY_SPI_SCB_CLK_PIN 2ul  /* Define the port settings */
#define CY_SPI_SCB_CLK_MUX P0_2_SCB0_SPI_CLK  /* Define the port settings */
#define CY_SPI_SCB_SEL0_PORT GPIO_PRT0  /* Define the port settings */
#define CY_SPI_SCB_SEL0_PIN 3ul  /* Define the port settings */
#define CY_SPI_SCB_SEL0_MUX P0_3_SCB0_SPI_SELECT0  /* Define the port settings */

#define CY_SPI_SCB_PCLK PCLK_SCB0_CLOCK  /* Define the peripheral clock */

#define CY_SPI_SCB_IRQN scb_0_interrupt_IRQn  /* Define the System interrupt index number */

/* Master Settings */
#define SCB_MISO_DRIVE_MODE CY_GPIO_DM_HIGHZ  /* Define the port settings */
#define SCB_MOSI_DRIVE_MODE CY_GPIO_DM_STRONG_IN_OFF  /* Define the port settings */
#define SCB_CLK_DRIVE_MODE CY_GPIO_DM_STRONG_IN_OFF  /* Define the port settings */
#define SCB_SEL0_DRIVE_MODE CY_GPIO_DM_STRONG_IN_OFF  /* Define the port settings */

/* User setting value */
#define SOURCE_CLOCK_FRQ 800000000ul
#define SCB_SPI_BAUDRATE 125000ul  /* Please set baudrate value of SPI you want */
#define SCB_SPI_OVERSAMPLING 16ul  /* Please set oversampling of SPI you want */  /* Define the clock parameters */
#define SCB_SPI_CLOCK_FREQ (SCB_SPI_BAUDRATE * SCB_SPI_OVERSAMPLING)

#define DIVIDER_NO_1 (1ul)

static cy_stc_gpio_pin_config_t SPI_port_pin_cfg = /* Configure the port setting parameters */
{
    .outVal = 0ul,
    .driveMode = 0ul,  /* Will be updated in runtime */
    .hsiom = HSIOM_SEL_GPIO,  /* Will be updated in runtime */
    .intEdge = 0ul,
    .intMask = 0ul,
    .vtrip = 0ul,
    .slewRate = 0ul,
    .driveSel = 0ul,
};

static cy_stc_sysint_irq_t irq_cfg =
```

Application note 11 002-25401 Rev. *E 2023-11-10
How to use serial communications block (SCB) in TRAVEO™ T2G family

3 SPI setting procedure example

```c
{
    .sysIntSrc = CY_SPI_SCB_IRQN,  /* Configure the interrupt structure parameters */
    .intIdx = CPUIntIdx3_IRQn,
    .isEnabled = true,
};
```

```c
uint16_t readData[2];
```

```c
static const cy_stc_scb_spi_config_t SCB_SPI_cfg = /* Configure the SCB parameters */
{
    .spiMode = CY_SCB_SPI_MASTER,    /* Specifies the mode of operation    */
    .subMode = CY_SCB_SPI_MOTOROLA,  /* Specifies the sub mode of SPI operation */
    .sclkMode = CY_SCB_SPI_CPHA0_CPOL0,   /* Clock is active low, data is changed on first edge */
    .oversample = SCB_SPI_OVERSAMPLING,   /* SPI_CLOCK divided by SCB_SPI_OVERSAMPLING should be baudrate */
    .rxDataWidth = 16ul,    /* The width of RX data (valid range 4-16). It must be the same as \ref txDataWidth except in National sub-mode. */
    .txDataWidth = 16ul,    /* The width of TX data (valid range 4-16). It must be the same as \ref rxDataWidth except in National sub-mode. */
    .enableMsbFirst = true,    /* Enables the hardware to shift out the data element MSB first, otherwise, LSB first */
    .enableFreeRunSclk = false,    /* Enables the master to generate a continuous SCLK regardless of whether there is data to send */
    .enableInputFilter = false,    /* Enables a digital 3-tap median filter to be applied to the input of the RX FIFO to filter glitches on the line. */
    .enableMisoLateSample = true,    /* Enables the master to sample MISO line one half clock later to allow better timings. */
    .enableTransferSeperation = true,    /* Enables the master to transmit each data element separated by a de-assertion of the slave select line (only applicable for the master mode) */
    .ssPolarity0 = false,    /* SS0: active low */
    .ssPolarity1 = false,    /* SS1: active low */
    .ssPolarity2 = false,    /* SS2: active low */
    .ssPolarity3 = false,    /* SS3: active low */
    .enableWakeFromSleep = false,    /* When set, the slave will wake the device when the slave select line becomes active. Note that not all SCBs support this mode. Consult the device datasheet to determine which SCBs support wake from deep sleep. */
    .rxFifoTriggerLevel = 1ul,    /* Interrupt occurs, when there are more entries of 2 in the RX FIFO */
    .rxFifoIntEnableMask = 1ul,    /* Bits set in this mask will allow events to cause an interrupt */
    .txFifoTriggerLevel = 0ul,    /* When there are fewer entries in the TX FIFO, then at this level the TX trigger output goes high. This output can be connected to a DMA channel through a trigger mux. Also, it controls the \ref CY_SCB_SPI_TX_TRIGGER interrupt source. */
    .txFifoIntEnableMask = 0ul,    /* Bits set in this mask allow events to cause an interrupt */
    .masterSlaveIntEnableMask = 0ul,    /* Bits set in this mask allow events to cause an interrupt */
    .enableSpiDoneInterrupt = false,
    .enableSpiBusErrorInterrupt = false,
};
```
/* Master schedule handler */
static void SCB_TickHandler(void)
{
    Cy_SCB_SPI_WriteArray(CY_SPI_SCB_TYPE,(void*)readData, 2ul);
}

static void SchedulerInit(void)
{
    Cy_SysTick_Init(CY_SYSTICK_CLOCK_SOURCE_CLK_CPU, CORE_CLOCK_FRQ / 10ul); // 100[ms]
    Cy_SysTick_SetCallback(0ul, SCB_TickHandler);
    Cy_SysTick_Enable();
}

int main(void)
{
    SystemInit();

    /********************************************************************/  
    /** Calculate divider setting for the SCB ***/  /* (1) Configure the clock */  
    /********************************************************************/  
    Cy_SysClk_PeriphAssignDivider(CY_SPI_SCB_PCLK, CY_SYSCLK_DIV_24_5_BIT, DIVIDER_NO_1); /*
    Configure the Peripheral Clock (See Code Listing 4) */
    SetPeripheFracDiv24_5(SCB_SPI_CLOCK_FREQ, SOURCE_CLOCK_FRQ, DIVIDER_NO_1); /* Configure
    the divider (See Code Listing 2) */
    Cy_SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_24_5_BIT, 1ul); /* Enable the divider (See
    Code Listing 6) */

    __enable_irq(); /* Enable global interrupts. */ /* (2) Enable global interrupt*1 */

    /********************************************************************/  
    /* De-initialization for peripherals */  
    /********************************************************************/  
    Cy_SCB_SPI_DeInit(CY_SPI_SCB_TYPE); /* De-Initialize the SCB if necessary (See Code
    Listing 7) */

    /********************************************************************/  
    /* Interrupt setting for SPI communication */  
    /********************************************************************/  
    Cy_SysInt_InitIRQ(&irq_cfg); /* (3) Set the parameters to interrupt structure*1 */
    Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, irqSCB); /* (4) Set the system interrupt
    handler*1 (See Code Listing 3) */
    NVIC_EnableIRQ(irq_cfg.intIdx); /* (5) Interrupt Enable*1 */
How to use serial communications block (SCB) in TRAVEO™ T2G family

3  SPI setting procedure example

/***************************************************************************/
/* Port Setting for SPI communication */ /* (6) Set the SPI port2 */
/***************************************************************************/
/* According to the HW environment to change SCB CH*/
SPI_port_pin_cfg.
driveMode = SCB_MISO_DRIVE_MODE; /* Change the driveMode and set the port
setting parameters */
SPI_port_pin_cfg.hsiom = CY_SPI_SCB_MISO_MUX; /* Change the driveMode and set the port
setting parameters */
Cy_GPIO.Pin_Init(CY_SPI_SCB_MISO_PORT, CY_SPI_SCB_MISO_PIN, &SPI_port_pin_cfg); /* Change
the driveMode and set the port setting parameters */

SPI_port_pin_cfg.driveMode = SCB_MOSI_DRIVE_MODE; /* Change the driveMode and set the port
setting parameters */
SPI_port_pin_cfg.hsiom = CY_SPI_SCB_MOSI_MUX; /* Change the driveMode and set the port
setting parameters */
Cy_GPIO.Pin_Init(CY_SPI_SCB_MOSI_PORT, CY_SPI_SCB_MOSI_PIN, &SPI_port_pin_cfg); /* Change
the driveMode and set the port setting parameters */

SPI_port_pin_cfg.driveMode = SCB_CLK_DRIVE_MODE; /* Change the driveMode and set the port
setting parameters */
SPI_port_pin_cfg.hsiom = CY_SPI_SCB_CLK_MUX; /* Change the driveMode and set the port
setting parameters */
Cy_GPIO.Pin_Init(CY_SPI_SCB_CLK_PORT, CY_SPI_SCB_CLK_PIN, &SPI_port_pin_cfg); /* Change the
driveMode and set the port setting parameters */

SPI_port_pin_cfg.driveMode = SCB_SEL0_DRIVE_MODE; /* Change the driveMode and set the port
setting parameters */
SPI_port_pin_cfg.hsiom = CY_SPI_SCB_SEL0_MUX; /* Change the driveMode and set the port
setting parameters */
Cy_GPIO.Pin_Init(CY_SPI_SCB_SEL0_PORT, CY_SPI_SCB_SEL0_PIN, &SPI_port_pin_cfg); /* Change the
driveMode and set the port setting parameters */
]

/***************************************************************************/
/* SCB initialization for SPI communication */
/***************************************************************************/
Cy_SCB_SPI_Init(CY_SPI_SCB_TYPE, &SCB_SPI_cfg, NULL); /* (7) Initialize SCB for SPI
communication (See Code Listing 8) */
Cy_SCB_SPISetActiveSlaveSelect(CY_SPI_SCB_TYPE, 0ul); /* Set the using cannel number (See
Code Listing 9) */
Cy_SCB_SPI_Enable(CY_SPI_SCB_TYPE); /* (8) Enable SPI (See Code Listing 10) */

/***************************************************************************/
/* Write initial value to buffer */
/***************************************************************************/
3 SPI setting procedure example

```c
readData[0] = 0xAAAAul; /* If necessary, Initialize the buffer values */
readData[1] = 0xAAAAul;

SchedulerInit(); /* (9) Initialize the scheduler */
for(;;);
```

*1: For details, see the CPU interrupt handing sections in the architecture TRM.

*2: For details, see the I/O System sections in the architecture TRM.

**Code Listing 2** lists the fractional clock divider function.

**Code Listing 2** SetPeripheFracDiv24_5() function

```c
void SetPeripheFracDiv24_5(uint64_t targetFreq, uint64_t sourceFreq, uint8_t divNum) /* Create
the function to determine the divider division ratio */
{
    uint64_t temp = ((uint64_t)sourceFreq << 5ul);
    uint32_t divSetting;

    divSetting = (uint32_t)(temp / targetFreq); /* Calculates the division ratio */
    Cy_SysClk_PeriphSetFracDivider(CY_SYSCLK_DIV_24_5_BIT, divNum,
                                 (((divSetting >> 5ul) & 0x00000FFfu) - 1ul),
                                 (divSetting & 0x000001Fu)); /* Set the division ratio
(See Code Listing 5) */
}
```

**Code Listing 3** demonstrates an example of the interrupt handler.
Code Listing 3 Interrupt handler example

```c
void irqSCB(void)
{
    uint32_t status;

    status = Cy_SCB_SPI_GetRxFifoStatus(CY_SPI_SCB_TYPE); /* (10) Check the Interrupt is Active (See Code Listing 11) */
    if(status & CY_SCB_SPI_RX_TRIGGER)
    {
        Cy_SCB_SPI_ReadArray(CY_SPI_SCB_TYPE, (void*)readData, 2ul); /* (11) Read the Received SPI Data (See Code Listing 12) */
        Cy_SCB_SPI_ClearRxFifoStatus(CY_SPI_SCB_TYPE, CY_SCB_SPI_RX_TRIGGER); /* (12) Clear the RX TRIGGER Interrupt Flag (See Code Listing 13) */
    }
}

/* Master schedule handler */
static void SCB_TickHandler(void) /* Interrupt handler for TX */
{
    Cy_SCB_SPI_WriteArray(CY_SPI_SCB_TYPE, (void*)readData, 2ul); /* (13) Write and send the Received SPI Data (See Code Listing 14) */
}
```

*1: For details, see the CPU interrupt handing sections in the architecture TRM.

Code Listing 4 to Code Listing 6 demonstrate an example program to configure CLK in the driver part. The following description will help you understand the register notation of the driver part of SDL:

- PERI->unCLOCK_CTL and unDIV is the PERI_CLOCK_CTLx register mentioned in the registers TRM.
- Performance improvement measuresFor register setting performance improvement, the SDL writes a complete 32-bit data to the register. Each bit field is generated in advance in a bit-writable buffer and written to the register as the final 32-bit data.

```c
un_PERI_CLOCK_CTL_t tempCLOCK_CTL_RegValue;tempCLOCK_CTL_RegValue.u32Register = PERI->unCLOCK_CTL[ipBlock].u32Register;tempCLOCK_CTL_RegValue.stcField.u2TYPE_SEL = dividerType;tempCLOCK_CTL_RegValue.stcField.u8DIV_SEL = dividerNum;PERI->unCLOCK_CTL[ipBlock].u32Register = tempCLOCK_CTL_RegValue.u32Register;
```
- See cy_sysclk.h under hdr/rev_x/ip for more information on the union and structure representation of registers.
Code Listing 4 Cy_SysClk_PeriphAssignDivider() function

```
__STATIC_INLINE cy_en_sysclk_status_t Cy_SysClk_PeriphAssignDivider(en_clk_dst_t ipBlock,
                        cy_en_divider_types_t dividerType, uint32_t dividerNum)
{
    if(Cy_SysClk_CheckDividerExisting(dividerType, dividerNum) == CY_DIVIDER_NOT_EXISTING) /*
      Check if configuration parameter values are valid */
    {
        return CY_SYSCLK_BAD_PARAM;
    }

    un_PERI_CLOCK_CTL_t tempCLOCK_CTL_RegValue;
    tempCLOCK_CTL_RegValue.u32Register = PERI->unCLOCK_CTL[ipBlock].u32Register;
    tempCLOCK_CTL_RegValue.stcField.u2TYPE_SEL = dividerType;
    tempCLOCK_CTL_RegValue.stcField.u8DIV_SEL = dividerNum;
    PERI->unCLOCK_CTL[ipBlock].u32Register = tempCLOCK_CTL_RegValue.u32Register;

    return CY_SYSCLK_SUCCESS;
}
```
Code Listing 5 Cy_SysClk_PeriphSetFracDivider() function

```c
__STATIC_INLINE cy_en_sysclk_status_t Cy_SysClk_PeriphSetFracDivider(cy_en_divider_types_t dividerType, uint32_t dividerNum, uint32_t dividerIntValue, uint32_t dividerFracValue)
{
    if(Cy_SysClk_CheckDividerExisting(dividerType, dividerNum) == CY_DIVIDER_NOT_EXISTING) /* Check if configuration parameter values are valid */
    {
        return CY_SYSCLK_BAD_PARAM;
    }
    if (dividerType == CY_SYSCLK_DIV_16_5_BIT)
    {
        if ((dividerIntValue <= (PERI_DIV_16_5_CTL_INT16_DIV_Msk >> PERI_DIV_16_5_CTL_INT16_DIV_Pos)) &&
            (dividerFracValue <= (PERI_DIV_16_5_CTL_FRAC5_DIV_Msk >> PERI_DIV_16_5_CTL_FRAC5_DIV_Pos)))
            if (dividerType == CY_SYSCLK_DIV_16_5_BIT)
            {
                if ((dividerIntValue <= (PERI_DIV_16_5_CTL_INT16_DIV_Msk >> PERI_DIV_16_5_CTL_INT16_DIV_Pos)) &&
                    (dividerFracValue <= (PERI_DIV_16_5_CTL_FRAC5_DIV_Msk >> PERI_DIV_16_5_CTL_FRAC5_DIV_Pos)))
                {
                    PERI->unDIV_16_5_CTL[dividerNum].stcField.u16INT16_DIV = dividerIntValue;
                    PERI->unDIV_16_5_CTL[dividerNum].stcField.u5FRAC5_DIV = dividerFracValue;
                }
            }
    }
    else if (dividerType == CY_SYSCLK_DIV_24_5_BIT) /* Check the dividerType */
    {
        if ((dividerIntValue <= (PERI_DIV_24_5_CTL_INT24_DIV_Msk >> PERI_DIV_24_5_CTL_INT24_DIV_Pos)) && (dividerFracValue <= (PERI_DIV_24_5_CTL_FRAC5_DIV_Msk >> PERI_DIV_24_5_CTL_FRAC5_DIV_Pos)))
        {
            PERI->unDIV_24_5_CTL[dividerNum].stcField.u24INT24_DIV = dividerIntValue; /* Select INT24_DIV bits */
            PERI->unDIV_24_5_CTL[dividerNum].stcField.u5FRAC5_DIV = dividerFracValue; /* Select FRAC5_DIV bits */
        }
    }
    else
    {
        return CY_SYSCLK_BAD_PARAM;
    }
}
```
Code Listing 6 Cy_SysClk_PeriphEnableDivider() function

```c
__STATIC_INLINE cy_en_sysclk_status_t Cy_SysClk_PeriphEnableDivider(cy_en_divider_types_t dividerType, uint32_t dividerNum)
{
    if(Cy_SysClk_CheckDividerExisting(dividerType, dividerNum) == CY_DIVIDER_NOT_EXISTING) /* Check if configuration parameter values are valid */
    {
        return CY_SYSCLK_BAD_PARAM;
    }

    /* specify the divider, make the reference = clk_peri, and enable the divider */
    un_PERI_DIV_CMD_t tempDIV_CMD_RegValue;
    tempDIV_CMD_RegValue.u32Register = PERI->unDIV_CMD.u32Register;  // Reference = clk_peri
    tempDIV_CMD_RegValue.stcField.u1ENABLE = 1ul; /* Enable the Divider */
    tempDIV_CMD_RegValue.stcField.u2PA_TYPE_SEL = 3ul;
    tempDIV_CMD_RegValue.stcField.u8PA_DIV_SEL = 0xFFul;
    tempDIV_CMD_RegValue.stcField.u2TYPE_SEL = dividerType;
    tempDIV_CMD_RegValue.stcField.u8DIV_SEL = dividerNum;
    PERI->unDIV_CMD.u32Register = tempDIV_CMD_RegValue.u32Register;

    (void)PERI->unDIV_CMD; /* dummy read to handle buffered writes */
    return CY_SYSCLK_SUCCESS;
}
```

**How to use serial communications block (SCB) in TRAVEO™ T2G family**

3 SPI setting procedure example

Code Listing 7 to Code Listing 14 demonstrate an example program to configure the SCB in the driver part. The following description will help you understand the register notation of the driver part of SDL:
### Code Listing 7 Cy_SCB_SPI_DeInit() function

```c
void Cy_SCB_SPI_DeInit(volatile stc_SCB_t *base)
{
    /* SPI interface */
    base->unCTRL.u32Register = CY_SCB_CTRL_DEF_VAL; /* Set the unCTRL reg */
    base->unSPI_CTRL.u32Register = CY_SCB_SPI_CTRL_DEF_VAL; /* Set the unSPI_CTRL Reg */

    /* RX direction */
    base->unRX_CTRL.u32Register = CY_SCB_RX_CTRL_DEF_VAL; /* Set the unRX_CTRL Reg */
    base->unRX_FIFO_CTRL.u32Register = 0ul; /* Set the unRX_FIFO_CTRL Reg to "0" */

    /* TX direction */
    base->unTX_CTRL.u32Register = CY_SCB_TX_CTRL_DEF_VAL; /* Set the unTX_CTRL Reg */
    base->unTX_FIFO_CTRL.u32Register = 0ul; /* Set the unTX_FIFO_CTRL Reg to "0" */

    /* Disable all interrupt sources */
    base->unINTR_SPI_EC_MASK.u32Register = 0ul; /* Disable the all interrupt */
    base->unINTR_I2C_EC_MASK.u32Register = 0ul; /* Disable the all interrupt */
    base->unINTR_RX_MASK.u32Register = 0ul; /* Disable the all interrupt */
    base->unINTR_TX_MASK.u32Register = 0ul; /* Disable the all interrupt */
    base->unINTR_M_MASK.u32Register = 0ul; /* Disable the all interrupt */
    base->unINTR_S_MASK.u32Register = 0ul; /* Disable the all interrupt */
}
```
Code Listing 8 Cy_SCB_SPI_Init() function

cy_en_scb_spi_status_t Cy_SCB_SPI_Init(volatile stc_SCB_t *base, cy_stc_scb_spi_config_t const *config, cy_stc_scb_spi_context_t *context)
{
    cy_en_scb_spi_status_t retStatus = CY_SCB_SPI_BAD_PARAM;
    un_SCB_CTRL_t     tscbCtrl   = { 0ul};
    un_SCB_SPI_CTRL_t tscbSpiCtrl = { 0ul};
    un_SCB_TX_CTRL_t  tscbTxCtrl  = { 0ul};
    un_SCB_RX_CTRL_t  tscbRxCtrl  = { 0ul};
    uint32_t          locSclkMode = 0ul;
    uint32_t          maxOfDataWidth = 0ul;

    if ((NULL != base) && (NULL != config)) /* Check if configuration parameter values are valid */
    {
        /* Set SCLK mode for TI - CY_SCB_SPI_CPHA1_CPOL0, NS - CY_SCB_SPI_CPHA0_CPOL0, Motorola - take from config */
        if(CY_SCB_SPI_MOTOROLA == config->subMode)
        {
            locSclkMode = config->sclkMode;
        }
        else if(CY_SCB_SPI_NATIONAL == config->subMode)
        {
            locSclkMode = CY_SCB_SPI_CPHA0_CPOL0;
        }
        else
        {
            locSclkMode = CY_SCB_SPI_CPHA1_CPOL0;
        }

        maxOfDataWidth = (config->rxDataWidth >= config->txDataWidth) ? config->rxDataWidth : config->txDataWidth;
        if ( maxOfDataWidth <= CY_SCB_BYTE_WIDTH )
        {
            tscbCtrl.stcField.u2MEM_WIDTH = CY_SCB_SPI_MEM_WIDTH_BYTE;
        }
        else if ( maxOfDataWidth <= CY_SCB_HALFWORD_WIDTH )
        {
            tscbCtrl.stcField.u2MEM_WIDTH = CY_SCB_SPI_MEM_WIDTH_HALFWORD;
        }
        else if ( maxOfDataWidth <= CY_SCB_WORD_WIDTH )
        {
            tscbCtrl.stcField.u2MEM_WIDTH = CY_SCB_SPI_MEM_WIDTH_WORD;
        }
        else
        {
            return CY_SCB_SPI_BAD_PARAM;
        }

        if ( config->enableWakeFromSleep ) /* Set the SPI communication parameters */
        {
            tscbCtrl.stcField.u1EC_AM_MODE = true;
        }
    }
}
else {
    tscbCtrl.stcField.u1EC_AM_MODE = false;
}

tscbCtrl.stcField.u4OVS = (config->oversample - 1ul); /* Set the SPI communication parameters */
tscbCtrl.stcField.u2MODE = CY_SCB_CTRL_MODE_SPI; /* Set the SPI communication parameters */

base->unCTRL.u32Register = tscbCtrl.u32Register; /* Set the SPI communication parameters */

tscbSpiCtrl.stcField.u1SSEL_CONTINUOUS = ~(config->enableTransferSeperation); /* Set the SPI communication parameters */
tscbSpiCtrl.stcField.u1SELECT_PRECEDE = (0ul != (CY_SCB_SPI_TI_PRECEDE & config->subMode) ? 1ul : 0ul); /* Set the SPI communication parameters */
tscbSpiCtrl.stcField.u1LATE_MISO_SAMPLE = config->enableMisoLateSample; /* Set the SPI communication parameters */
tscbSpiCtrl.stcField.u1SCLK_CONTINUOUS = config->enableFreeRunSclk; /* Set the SPI communication parameters */
tscbSpiCtrl.stcField.u1MASTER_MODE = ((CY_SCB_SPI_MASTER == config->spiMode) ? 1ul : 0ul); /* Set the SPI communication parameters */
tscbSpiCtrl.stcField.u1CPHA = ((locSclkMode >> 1ul) & 0x01ul); /* Set the SPI communication parameters */
tscbSpiCtrl.stcField.u1CPOL = (locSclkMode & 0x01ul); /* Set the SPI communication parameters */
tscbSpiCtrl.stcField.u1SSEL_POLARITY0 = config->ssPolarity0; /* Set the SPI communication parameters */
tscbSpiCtrl.stcField.u1SSEL_POLARITY1 = config->ssPolarity1; /* Set the SPI communication parameters */
tscbSpiCtrl.stcField.u1SSEL_POLARITY2 = config->ssPolarity2; /* Set the SPI communication parameters */
tscbSpiCtrl.stcField.u1SSEL_POLARITY3 = config->ssPolarity3; /* Set the SPI communication parameters */
tscbSpiCtrl.stcField.u2MODE = config->subMode; /* Set the SPI communication parameters */

base->unSPI_CTRL.u32Register = tscbSpiCtrl.u32Register; /* Set the SPI communication parameters */

base->unRX_CTRL.u32Register = tscbRxCtrl.u32Register; /* Set the SPI communication parameters */

tscbRxCtrl.stcField.u1MSB_FIRST = config->enableMsbFirst; /* Set the SPI communication parameters */
tscbRxCtrl.stcField.u1MEDIAN = config->enableInputFilter; /* Set the SPI communication parameters */
tscbRxCtrl.stcField.u5DATA_WIDTH = (config->rxDataWidth - 1ul); /* Set the SPI communication parameters */

base->unRX_CTRL.u32Register = tscbRxCtrl.u32Register; /* Set the SPI communication parameters */

base->unRX_FIFO_CTRL.stcField.u8TRIGGER_LEVEL = config->rxFifoTriggerLevel; /* Set the SPI communication parameters */

tscbTxCtrl.stcField.u1MSB_FIRST = config->enableMsbFirst; /* Set the SPI communication parameters */
3 SPI setting procedure example

```c

/* SPI communication parameters */
tscbTxCtrl.stcField.u5DATA_WIDTH = (config->txDataWidth - 1ul); /* Set the SPI communication parameters */
base->unTX_CTRL.u32Register = tscbTxCtrl.u32Register; /* Set the SPI communication parameters */
base->unTX_FIFO_CTRL.stcField.u8TRIGGER_LEVEL = config->txFifoTriggerLevel; /* Set the SPI communication parameters */

/* Set up interrupt sources */
base->unINTR_TX_MASK.u32Register = config->txFifoIntEnableMask; /* Set the SPI interrupt */
base->unINTR_RX_MASK.u32Register = config->rxFifoIntEnableMask; /* Set the SPI interrupt */
base->unINTR_M.stcField.u1SPI_DONE = config->enableSpiDoneInterrupt; /* Set the SPI interrupt */
base->unINTR_S.stcField.u1SPI_BUS_ERROR = config->enableSpiBusErrorInterrupt; /* Set the SPI interrupt */

/* Initialize the context */
if (NULL != context) {
    context->status = 0UL; /* Set the TX/RX Buffer */
    context->txBufIdx = 0UL; /* Set the TX/RX Buffer */
    context->rxBufIdx = 0UL; /* Set the TX/RX Buffer */
    context->cbEvents = NULL; /* Set the TX/RX Buffer */

    #if !(NDEBUG)
        /* Put an initialization key into the initKey variable to verify * context initialization in the transfer API. */
        context->initKey = CY_SCB_SPI_INIT_KEY;
    #endif /* !(NDEBUG) */
}

    retStatus = CY_SCB_SPI_SUCCESS;
}

return (retStatus);
```

**Code Listing 9 Cy_SCB_SPI_SetActiveSlaveSelect() function**

```c
__STATIC_INLINE void Cy_SCB_SPI_SetActiveSlaveSelect(volatile stc_SCB_t *base, uint32_t slaveSelect) {
    base->unSPI_CTRL.stcField.u2SSEL = slaveSelect; /* Set the slave mode */
}
```
3.2 Slave mode

This example sets the Motorola SPI Slave mode so that the Master transmits two half-words of data to the Slave, and then the Slave receives two half-words of data from the Master.
3.2.1 Use case

- SCB mode = Motorola SPI Slave mode
- SCB channel = 1
- PCLK = 4 MHz
- Bit rate = 1 Mbps
- Tx/Rx data length = 16 bits
- Tx/Rx FIFO = Used (16-bit FIFO data elements)
- Tx/Rx interrupts = Enable

Used ports
- SCLK : SCB1_CLK (P18.2)
- MOSI : SCB1_MOSI (P18.1)  MOSI data is driven on a falling edge of SCLK.
- MISO : SCB1_MISO (P18.0)  MISO data is captured on a falling edge of SCLK after half a SPI SCLK period from a rising edge.
- SELECT : SCB1_SEL0 (P18.3)

Figure 4 shows the example of a connection between the SCB and another SPI device.

![Figure 4](image_url)

**Figure 4** Example of SPI (Slave mode) communication connection

SPI slave mode operation

In SPI mode, SCLK, MOSI, MISO, and SELECT signals connect to another SPI Master device. In Slave mode, SCLK, MOSI, and SELECT are input ports, and MISO is the output port. SELECT indicates when valid data is transmitted from the SPI Master device or SPI Slave device.

Figure 5 shows the setting procedure and operation example for Slave mode.
3 SPI setting procedure example

Figure 5  SPI slave mode operation

1. Configure the clock.
2. Enable global interrupt (CPU interrupt enable). For details, see the CPU interrupt handing sections in the architecture TRM.
3. Set the interrupt structure. For details, see the CPU interrupt handing sections in the architecture TRM.
4. Set the system interrupt handler. For details, see the CPU interrupt handing sections in the architecture TRM.
5. Enable the interrupt.
6. Set the SPI port for Slave mode. SCLK, MOSI, and SELECT are output. MISO is input.
7. Initialize the SCB for SPI communication.
8. Enable SPI.
9. When the SCB receives data, an Rx FIFO interrupt occurs.
10. The software reads the received data from the Rx FIFO.
11. Clear the Rx FIFO interrupt.
12. When the SCB transmits data, an Tx FIFO interrupt occurs.
13. The software writes and sends the received SPI data.
### Configuration and example

Table 4 lists the parameters of the configuration part in SDL for SPI Slave mode.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCE_CLOCK_FRQ</td>
<td>Input divider clock frequency</td>
<td>80000000ul (80MHz)</td>
</tr>
<tr>
<td>SCB_SPI_BAUDRATE</td>
<td>SPI baud rate</td>
<td>125000ul</td>
</tr>
<tr>
<td>SCB_SPI_OVERSAMPLING</td>
<td>Oversampling for SPI</td>
<td>16ul</td>
</tr>
<tr>
<td>SCB_SPI_CLOCK_FREQ</td>
<td>Peripheral clock frequency</td>
<td>SCB_SPI_BAUDRATE<em>SCB_SPI_OVERSAMPLING (125000</em>16 = 2 MHz)</td>
</tr>
<tr>
<td>DIVIDER_NO_1</td>
<td>Divider number</td>
<td>1ul</td>
</tr>
<tr>
<td>CY_SPI_SCB_PCLK</td>
<td>Peripheral clock number</td>
<td>PCLK_SCB0_CLOCK</td>
</tr>
<tr>
<td>CY_SPI_SCB_TYPE</td>
<td>SCB channel number</td>
<td>SCB0</td>
</tr>
<tr>
<td>SCB_SPI_cfg.spiMode</td>
<td>Operation mode</td>
<td>CY_SCB_SPI_SLAVE (0x0)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.subMode</td>
<td>SPI operation sub mode</td>
<td>CY_SCB_SPI_MOTOROLA (0x0)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.sclkMode</td>
<td>Clock polarity and phase</td>
<td>CY_SCB_SPI_CPHA0_CPOL0 (0x0)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.oversample</td>
<td>Oversampling factor</td>
<td>SCB_SPI_OVERSAMPLING (16ul)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.rxDataWidth</td>
<td>Rx data frame width</td>
<td>16ul</td>
</tr>
<tr>
<td>SCB_SPI_cfg.txDataWidth</td>
<td>Tx data frame width</td>
<td>16ul</td>
</tr>
<tr>
<td>SCB_SPI_cfg.enableMsbFirst</td>
<td>Least or most significant bit first</td>
<td>true (MSB)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.enableFreeRunSclk</td>
<td>SCLK generated mode</td>
<td>false</td>
</tr>
<tr>
<td>SCB_SPI_cfg.enableInputFilter</td>
<td>Median filter</td>
<td>false</td>
</tr>
<tr>
<td>SCB_SPI_cfg.enableMisoLateSample</td>
<td>SCLK edge on which MISO is captured</td>
<td>true (determined by CPOL and CPHA)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.enableTransferSeperatio</td>
<td>Continuous SPI data transfers</td>
<td>true (enable)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.ssPolarity0</td>
<td>Polarity of SELECT 1</td>
<td>false (‘0’ active, used)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.ssPolarity1</td>
<td>Polarity of SELECT 2</td>
<td>false (‘0’ active, unused)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.ssPolarity2</td>
<td>Polarity of SELECT 3</td>
<td>false (‘0’ active, unused)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.ssPolarity3</td>
<td>Polarity of SELECT 4</td>
<td>false (‘0’ active, unused)</td>
</tr>
<tr>
<td>SCB_SPI_cfg.enableWakeFromSleep</td>
<td>Slave selection detection logic</td>
<td>false</td>
</tr>
<tr>
<td>SCB_SPI_cfg.rxFifoTriggerLevel</td>
<td>Trigger level of Rx FIFO</td>
<td>1ul</td>
</tr>
<tr>
<td>SCB_SPI_cfg.rxFifoIntEnableMask</td>
<td>Interrupt of Rx FIFO</td>
<td>1ul</td>
</tr>
<tr>
<td>SCB_SPI_cfg.txFifoTriggerLevel</td>
<td>Trigger level of Tx FIFO</td>
<td>1ul</td>
</tr>
<tr>
<td>SCB_SPI_cfg.txFifoIntEnableMask</td>
<td>Interrupt of Tx FIFO</td>
<td>1ul</td>
</tr>
<tr>
<td>SCB_SPI_cfg.enableSpiDoneInterrupt</td>
<td>SPI master transfer done event</td>
<td>false</td>
</tr>
</tbody>
</table>

(table continues...)
### Table 4 (continued) List of SPI Slave mode configuration parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCB_SPI_cfg.enableSpiBusErrorInterrupt</td>
<td>SPI slave deselected at an unexpected time in the SPI transfer</td>
<td>false</td>
</tr>
<tr>
<td>CY_SPI_SCB_MISO_PORT</td>
<td>I/O output port number</td>
<td>GPIO_PRT0</td>
</tr>
<tr>
<td>CY_SPI_SCB_MISO_PIN</td>
<td>I/O output pin number</td>
<td>0ul</td>
</tr>
<tr>
<td>CY_SPI_SCB_MISO_MUX</td>
<td>Peripheral connection</td>
<td>P0_0_SCB0_SPI_MISO (30ul)</td>
</tr>
<tr>
<td>CY_SPI_SCB_MOSI_PORT</td>
<td>I/O output port number</td>
<td>GPIO_PRT0</td>
</tr>
<tr>
<td>CY_SPI_SCB_MOSI_PIN</td>
<td>I/O output pin number</td>
<td>1ul</td>
</tr>
<tr>
<td>CY_SPI_SCB_MOSI_MUX</td>
<td>Peripheral connection</td>
<td>P0_1_SCB0_SPI_MOSI (30ul)</td>
</tr>
<tr>
<td>CY_SPI_SCB_CLK_PORT</td>
<td>I/O output port number</td>
<td>GPIO_PRT0</td>
</tr>
<tr>
<td>CY_SPI_SCB_CLK_PIN</td>
<td>I/O output pin number</td>
<td>2ul</td>
</tr>
<tr>
<td>CY_SPI_SCB_CLK_MUX</td>
<td>Peripheral connection</td>
<td>P0_2_SCB0_SPI_CLK</td>
</tr>
<tr>
<td>CY_SPI_SCB_SEL0_PORT</td>
<td>I/O output port number</td>
<td>GPIO_PRT0</td>
</tr>
<tr>
<td>CY_SPI_SCB_SEL0_PIN</td>
<td>I/O output pin number</td>
<td>3ul</td>
</tr>
<tr>
<td>CY_SPI_SCB_SEL0_MUX</td>
<td>Peripheral connection</td>
<td>P0_3_SCB0_SPI_SELECT0</td>
</tr>
<tr>
<td>SCB_MISO_DRIVE_MODE</td>
<td>DRIVE_MODE for MISO</td>
<td>CY_GPIO_DM_STRONG_IN_OFF (0x06)</td>
</tr>
<tr>
<td>SCB_MOSI_DRIVE_MODE</td>
<td>DRIVE_MODE for MOSI</td>
<td>CY_GPIO_DM_HIGHZ (0x08)</td>
</tr>
<tr>
<td>SCB_CLK_DRIVE_MODE</td>
<td>DRIVE_MODE for CLK</td>
<td>CY_GPIO_DM_HIGHZ (0x08)</td>
</tr>
<tr>
<td>SCB_SEL0_DRIVE_MODE</td>
<td>DRIVE_MODE for SEL0</td>
<td>CY_GPIO_DM_HIGHZ (0x08)</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.outVal</td>
<td>Pin output state</td>
<td>0ul</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.driveMode</td>
<td>GPIO drive mode</td>
<td>0ul</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.hsiom</td>
<td>Connection for IO pin route</td>
<td>HSIOM_SEL_GPIO (0x0)</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.intEdge</td>
<td>Edge which will trigger an IRQ</td>
<td>0ul</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.intMask</td>
<td>Masks edge interrupt</td>
<td>0ul</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.vtrip</td>
<td>Input buffer mode</td>
<td>0ul</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.slewRate</td>
<td>Slew rate</td>
<td>0ul</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.driveSel</td>
<td>GPIO drive strength</td>
<td>0ul</td>
</tr>
<tr>
<td>CY_SPI_SCB_IRQN</td>
<td>System interrupt index number</td>
<td>scb_0_interrupt_IRQn (IDX: 17)</td>
</tr>
<tr>
<td>irq_cfg.sysIntSrc</td>
<td>System interrupt index number</td>
<td>CY_SPI_SCB_IRQN</td>
</tr>
<tr>
<td>irq_cfg.intIdx</td>
<td>CPU interrupt number</td>
<td>CPUIntIdx3_IRQn</td>
</tr>
<tr>
<td>.isEnabled</td>
<td>CPU interrupt enable</td>
<td>true (enable)</td>
</tr>
</tbody>
</table>

Table 5 lists the SPI parameters of the driver part in the SDL.

---

**How to use serial communications block (SCB) in TRAVEO™ T2G family**

3  **SPI setting procedure example**
### Table 5 List of functions

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cy_SCB_SPI_GetTxFifoStatus (volatile stc_SCB_t const *base)</td>
<td>Returns the current status of the Tx FIFO</td>
<td>*base: CY_SPI_SCB_TYPE</td>
</tr>
<tr>
<td>Cy_SCB_SPI_ClearTxFifoStatus (volatile stc_SCB_t *base, uint32_t clearMask)</td>
<td>Clears the selected statuses of the Tx FIFO</td>
<td>*base: CY_SPI_SCB_TYPE, clearMask: CY_SCB_SPI_TX_TRIGGER</td>
</tr>
</tbody>
</table>

**Code Listing 15 Example to configure SPI Slave mode in configuration part** demonstrates an example to configure SPI master mode in the configuration part.
Code Listing 15 Example to configure SPI Slave mode in configuration part

```c
/* Device Specific Settings */
#define CY_SPI_SCB_TYPE SCB0 /* Define the SCB channel */

#define CY_SPI_SCB_MISO_PORT GPIO_PRT0 /* Define the port settings */
#define CY_SPI_SCB_MISO_PIN 0ul /* Define the port settings */
#define CY_SPI_SCB_MISO_MUX P0_0_SCB0_SPI_MISO /* Define the port settings */
#define CY_SPI_SCB_MOSI_PORT GPIO_PRT0 /* Define the port settings */
#define CY_SPI_SCB_MOSI_PIN 1ul /* Define the port settings */
#define CY_SPI_SCB_MOSI_MUX P0_1_SCB0_SPI_MOSI /* Define the port settings */
#define CY_SPI_SCB_CLK_PORT GPIO_PRT0 /* Define the port settings */
#define CY_SPI_SCB_CLK_PIN 2ul /* Define the port settings */
#define CY_SPI_SCB_CLK_MUX P0_2_SCB0_SPI_CLK /* Define the port settings */
#define CY_SPI_SCB_SEL0_PORT GPIO_PRT0 /* Define the port settings */
#define CY_SPI_SCB_SEL0_PIN 3ul /* Define the port settings */
#define CY_SPI_SCB_SEL0_MUX P0_3_SCB0_SPI_SELECT /* Define the port settings */

#define CY_SPI_SCB_PCLK PCLK_SCB0_CLOCK /* Define the peripheral clock */
#define CY_SPI_SCB_IRQN scb_0_interrupt_IRQn /* Define the System interrupt index number */

/* Slave Settings */
#define SCB_MISO_DRIVE_MODE CY_GPIO_DM_STRONG_IN_OFF /* Define the port settings */
#define SCB_MOSI_DRIVE_MODE CY_GPIO_DM_HIGHZ /* Define the port settings */
#define SCB_CLK_DRIVE_MODE CY_GPIO_DM_HIGHZ /* Define the port settings */
#define SCB_SEL0_DRIVE_MODE CY_GPIO_DM_HIGHZ /* Define the port settings */

/* User setting value */
#define SOURCE_CLOCK_FRQ 80000000ul /*Define the clock parameters */
#define SCB_SPI_BAUDRATE 125000ul /* Please set baudrate value of SPI you want */ /*Define the clock parameters */
#define SCB_SPI_OVERSAMPLING 16ul /* Please set oversampling of SPI you want */ /*Define the clock parameters */
#define SCB_SPI_CLOCK_FREQ (SCB_SPI_BAUDRATE * SCB_SPI_OVERSAMPLING) /*Define the clock parameters */

#define DIVIDER_NO_1 (1ul)

static cy_stc_gpio_pin_config_t SPI_port_pinCfg = /* Configure the port setting parameters */ {
    .outVal = 0ul,
    .driveMode = 0ul, /* Will be updated in runtime */
    .hsiom = HSIOM_SEL_GPIO, /* Will be updated in runtime */
    .intEdge = 0ul,
    .intMask = 0ul,
    .vtrip = 0ul,
    .slewRate = 0ul,
};
```

How to use serial communications block (SCB) in TRAVEO™ T2G family

3  SPI setting procedure example
static cy_stc_sysint_irq_t irq_cfg = /* Configure the interrupt structure parameters */ {  .sysIntSrc = CY_SPI_SCB_IRQN,  .intIdx = CPUIntIdx3_IRQn,  .isEnabled = true,};
uint16_t readData[2];
uint16_t initialWriteData[4] = {0x1122u, 0x3344u, 0x5566u, 0x7788u};
#define SIZE_OF_INITIAL_DATA (sizeof(initialWriteData)/sizeof(uint16_t))
static const cy_stc_scb_spi_config_t SCB_SPI_cfg = /* Configure the SCB parameters */ {  .spiMode = CY_SCB_SPI_SLAVE, /* Specified the mode of operation */  .subMode = CY_SCB_SPI_MOTOROLA, /* Specified the submode of SPI operation */  .sclkMode = CY_SCB_SPI_CPHA0_CPOL0, /* Clock is active low, data is changed on first edge */  .oversample = SCB_SPI_OVERSAMPLING, /* SPI_CLOCK divided by SCB_SPI_OVERSAMPLING should be baudrate */  .rxDataWidth = 16ul, /* The width of RX data (valid range 4-16). It must be the same as \ref txDataWidth except in National sub-mode. */  .txDataWidth = 16ul, /* The width of TX data (valid range 4-16). It must be the same as \ref rxDataWidth except in National sub-mode. */  .enableMsbFirst = true, /* Enables the hardware to shift out the data element MSB first, otherwise, LSB first */  .enableFreeRunSclk = false, /* Enables the master to generate a continuous SCLK regardless of whether there is data to send */  .enableInputFilter = false, /* Enables a digital 3-tap median filter to be applied to the input of the RX FIFO to filter glitches on the line. */  .enableMisoLateSample = true, /* Enables the master to sample MISO line one half clock later to allow better timings. */  .enableTransferSeperation = true, /* Enables the master to transmit each data element separated by a de-assertion of the slave select line (only applicable for the master mode) */  .ssPolarity0 = false, /* SS0: active low */  .ssPolarity1 = false, /* SS1: active low */  .ssPolarity2 = false, /* SS2: active low */  .ssPolarity3 = false, /* SS3: active low */  .enableWakeFromSleep = false, /* When set, the slave will wake the device when the slave select line becomes active. Note that not all SCBs support this mode. Consult the device datasheet to determine which SCBs support wake from deep sleep. */  .rxFifoTriggerLevel = 1ul, /* Interrupt occurs, when there are more entries of 2 in the RX FIFO */  .txFifoTriggerLevel = 1ul, /* When there are fewer entries in the TX FIFO, then at this level the TX trigger output goes high. This output can be connected to a DMA channel through a trigger mux. Also, it controls the \ref CY_SCB_SPI_TX_TRIGGER interrupt source. */ /
/** Bits set in this mask allow events to cause an interrupt */

/* Create the function to determine the divider division ratio */

void SetPeripheFracDiv24_5(uint64_t targetFreq, uint64_t sourceFreq, uint8_t divNum)

{ /* Calculates the division ratio */

    uint64_t temp = ((uint64_t)sourceFreq << 5ul);
    uint32_t divSetting;

    divSetting = (uint32_t)(temp / targetFreq);

    Cy_SysClk_PeriphSetFracDivider(CY_SYSCLK_DIV_24_5_BIT, divNum,
        (((divSetting >> 5ul) & 0x00000FFFul) - 1ul),
        (divSetting & 0x0000001Ful)); /* Set the division ratio */

    (See Code Listing 5) */
}

int main(void)

{ /* Configure the clock */

    SystemInit();

    /******************************************************************************/
    /******************************************************************************/

    Cy_SysClk_PeriphAssignDivider(CY_SPI_SCB_PCLK, CY_SYSCLK_DIV_24_5_BIT, DIVIDER_NO_1);/*
    Configure the Peripheral Clock (See Code Listing 4) */

    SetPeripheFracDiv24_5(SCB_SPI_CLOCK_FREQ, SOURCE_CLOCK_FRQ, DIVIDER_NO_1); /* Configure
    the divider (See Code Listing 2) */

    Cy_SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_24_5_BIT, 1ul); /* Enable the divider (See
    Code Listing 6) */

    __enable_irq(); /* Enable global interrupts. */ /* Enable global interrupt */

    /*******************************************************************************/
    /******************************************************************************/

    De-initialization for peripherals */
    /*******************************************************************************/

    Cy_SCB_SPI_DeInit(CY_SPI_SCB_TYPE); /* De-Initialize the SCB if necessary (See
    Code Listing 7) */
How to use serial communications block (SCB) in TRAVEO™ T2G family

3 SPI setting procedure example

/***********************************/
/* Interrupt setting for SPI communication */
/***********************************/
Cy_SysInt_InitIRQ(&irq_cfg); /* (3) Set the parameters to interrupt structure*1 */
Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, irqSCB); /* (4) Set the system interrupt handler*1 (See Code Listing 3) */
NVIC_EnableIRQ(irq_cfg.intIdx); /* (5) Interrupt Enable*1 */

/***********************************/
/* Port Setting for SPI communication */
/* (6) Set the SPI port*2 */
/***********************************/
/* According to the HW environment to change SCB CH*/

SPI_port_pin_cfg.driveMode = SCB_MISO_DRIVE_MODE; /* Change the driveMode and set the port setting parameters */
SPI_port_pin_cfg.hsio = CY_SPI_SCB_MISO_MUX; /* Change the driveMode and set the port setting parameters */
Cy_GPIO_Pin_Init(CY_SPI_SCB_MISO_PORT, CY_SPI_SCB_MISO_PIN, &SPI_port_pin_cfg); /* Change the driveMode and set the port setting parameters */

SPI_port_pin_cfg.driveMode = SCB_MOSI_DRIVE_MODE; /* Change the driveMode and set the port setting parameters */
SPI_port_pin_cfg.hsio = CY_SPI_SCB_MOSI_MUX; /* Change the driveMode and set the port setting parameters */
Cy_GPIO_Pin_Init(CY_SPI_SCB_MOSI_PORT, CY_SPI_SCB_MOSI_PIN, &SPI_port_pin_cfg); /* Change the driveMode and set the port setting parameters */

SPI_port_pin_cfg.driveMode = SCB_CLK_DRIVE_MODE; /* Change the driveMode and set the port setting parameters */
SPI_port_pin_cfg.hsio = CY_SPI_SCB_CLK_MUX; /* Change the driveMode and set the port setting parameters */
Cy_GPIO_Pin_Init(CY_SPI_SCB_CLK_PORT, CY_SPI_SCB_CLK_PIN, &SPI_port_pin_cfg); /* Change the driveMode and set the port setting parameters */

SPI_port_pin_cfg.driveMode = SCB_SEL0_DRIVE_MODE; /* Change the driveMode and set the port setting parameters */
SPI_port_pin_cfg.hsio = CY_SPI_SCB_SEL0_MUX; /* Change the driveMode and set the port setting parameters */
Cy_GPIO_Pin_Init(CY_SPI_SCB_SEL0_PORT, CY_SPI_SCB_SEL0_PIN, &SPI_port_pin_cfg); /* Change the driveMode and set the port setting parameters */

/*************************************/
/* SCB initialization for SPI communication */
How to use serial communications block (SCB) in TRAVEO™ T2G family

3 SPI setting procedure example

```c
/*****************************/
Cy_SCB_SPI_Init(CY_SPI_SCB_TYPE, &SCB_SPI_cfg, NULL); /* (7) Initialize SCB for SPI communication (See Code Listing 8) */
Cy_SCB_SPISetActiveSlaveSelect(CY_SPI_SCB_TYPE, 0ul); /* Set the using channel number (See Code Listing 9) */
Cy_SCB_SPI_Enable(CY_SPI_SCB_TYPE); /* (8) Enable SPI (See Code Listing 10) */

for(;;);
}
```

*1: For details, see the CPU interrupt handing sections in the architecture TRM.
*2: For details, see the I/O System sections in the architecture TRM.

**Code Listing 16 Interrupt handler example**

```c
void irqSCB(void)
{
    uint32_t status;

    status = Cy_SCB_SPI_GetRxFifoStatus(CY_SPI_SCB_TYPE); /* (9) Check the Interrupt is Active (See Code Listing 11) */
    if(status & CY_SCB_SPI_RX_TRIGGER)
    {
        /*** Read data from RX FIFO ***/
        Cy_SCB_SPI_ReadArray(CY_SPI_SCB_TYPE, (void*)readData, 2ul); /* (10) Read the Received SPI Data (See Code Listing 12) */
        Cy_SCB_SPI_ClearRxFifoStatus(CY_SPI_SCB_TYPE, CY_SCB_SPI_RX_TRIGGER); /* (11) Clear the RX TRIGGER Interrupt Flag (See Code Listing 13) */
    }

    status = Cy_SCB_SPI_GetTxFifoStatus(CY_SPI_SCB_TYPE); /* (12) Check the TX Interrupt is Active (See Code Listing 17) */
    if(status & CY_SCB_SPI_TX_TRIGGER)
    {
        /*** Write back the data to TX FIFO ***/
        Cy_SCB_SPI_WriteArray(CY_SPI_SCB_TYPE, (void*)initialWriteData, SIZE_OF_INITIAL_DATA); /* (13) Write and send SPI Data (See Code Listing 14) */
        Cy_SCB_SPI_ClearTxFifoStatus(CY_SPI_SCB_TYPE, CY_SCB_SPI_TX_TRIGGER); /* (14) Clear the TX TRIGGER interrupt flag (See Code Listing 18) */
    }
}
```

*1: For details, see the CPU interrupt handing sections in the architecture TRM.

**Code Listing 17** and **Code Listing 18** demonstrate an example program to configure the SCB in the driver part. The following description will help you understand the register notation of the driver part of SDL:
Code Listing 17 Cy_SCB_SPI_GetTx FifoStatus() function

```c
__STATIC_INLINE uint32_t Cy_SCB_SPI_GetTxFifoStatus(volatile stc_SCB_t const *base) {
    return (Cy_SCB_GetTxInterruptStatus(base) & CY_SCB_SPI_TX_INTR); /* Read and check the Tx Interrupt */
}
```

Code Listing 18 Cy_SCB_SPI_ClearTxFifoStatus() function

```c
__STATIC_INLINE void Cy_SCB_SPI_ClearTxFifoStatus(volatile stc_SCB_t *base, uint32_t clearMask) {
    Cy_SCB_ClearTxInterrupt(base, clearMask); /* Clear the Tx Interrupt Factor */
}
```
4 UART setting procedure example

This section shows an example of SPI using the sample driver library (SDL). The SCB supports SPI Master mode and SPI Slave mode with Motorola, Texas Instruments, and National Semiconductor protocols. See the architecture technical TRM for details of each protocol. The code snippets in this application note are part of SDL, and is based on the CYT2B7 series. See Other references for the SDL.

The SCB features standard UART and multi-processor mode, SmartCard (ISO7816) reader, IrDA, and LIN (Slave mode). See the architecture technical TRM for details of each protocol. In this section, the procedure to set standard UART is explained as an example.

4.1 UART mode

This sample shows the usage of the SCB in standard UART mode. In this use case, after the respective registers are configured, the SCB transmits one byte of data to another device, and waits for an Rx data from another device.

4.1.1 Use case

- SCB mode = Standard UART
- SCB channel = 3
- PCLK = 80 MHz
- Baud rate = 115,200 bps

[Baud rate setting]

The baud rate calculation formula is as follows:

Baud rate [bps] = Input clock [Hz] / OVS

OVS: SCB_CTRL.OVS + 1

For example, the following shows how to calculate a real UART baud rate from an ideal UART baud rate:

- CLK_PERI frequency = 40 [MHz]
- target UART baud rate(Bit rate) = 115,200 [bps]
- OVS = 16 [oversamples]

You can use the specified CLK_PERI frequency, target UART baud rate, and OVS for calculating the real baud rate.

First, the ideal input clock to SCB is calculated:

Ideal input clock = Target baud rate * OVS = 115,200 * 16 = 1,843,200 [Hz]

Next, the ideal value of the clock divider control register (DIV24.5) required can be calculated:

Ideal DIV24.5 = 40 [MHz] / 1,843,200 [Hz] = 21.7014

However, the DIV24.5 register has 24 bits for the integer part and limited 5 bits for the fraction part (based 1/32). Therefore, the real divider value and the real UART baud rate can be calculated as follows:

Real DIV24.5 = 21.6875 (integer: 21, fractional: 22/32)

Real UART baud rate = 40 [MHz] / 21.6875 / 16 = 115,274 [bps]

For more details, see the architecture TRM.

- Data width = 8 bits
- Parity = None
- Stop bits = 1
- Flow control = None
- Tx/Rx FIFO = Used
- Rx interrupt = Disable
How to use serial communications block (SCB) in TRAVEO™ T2G family

4 UART setting procedure example

- Used ports
  - Tx : SCB3_TX (P13.1)
  - Rx : SCB3_RX (P13.0)

- The MCU receives the data transmitted by the UART device. The MCU sends the received data as it is. The initial message and data sent by the MCU are displayed on the PC.

An example Tx-Rx connection between the SCB and the external UART device is shown in Figure 6. In this example, flow control signals RTS and CTS are not used.

Figure 6 Example of UART communication connection

Figure 7 shows the setting procedure and operation example for the UART.

Figure 7 UART operation

(1) Set the UART port.
(2) Initialize SCB for UART.
(3) Enable UART.
(4) Configure the clock.
(5) Read the data from Rx FIFO.
(6) Write the data to Tx FIFO.

4.1.2 Configuration and example

Table 6 lists the parameters of the configuration part in the SDL for UART mode.
### Table 6  List of UART mode configuration parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_OVERSAMPLING</td>
<td>Oversampling for UART</td>
<td>8ul</td>
</tr>
<tr>
<td>CY_USB_SCB_UART_PCLK</td>
<td>Peripheral clock number</td>
<td>PCLK_SCB3_CLOCK (17ul)</td>
</tr>
<tr>
<td>g_stc_uart_config.uartMode</td>
<td>Submode of UART operation</td>
<td>CY_SCB_UART_STANDARD (0ul)</td>
</tr>
<tr>
<td>g_stc_uart_config.oversample</td>
<td>Oversampling for UART</td>
<td>UART_OVERSAMPLING (8ul)</td>
</tr>
<tr>
<td>g_stc_uart_config.dataWidth</td>
<td>Dataframe width</td>
<td>8ul</td>
</tr>
<tr>
<td>g_stc_uart_config.enableMsbFirst</td>
<td>LSB first or MSB first</td>
<td>False (LSB)</td>
</tr>
<tr>
<td>g_stc_uart_config.stopBits</td>
<td>Stop bit</td>
<td>CY_SCB_UART_STOP_BITS_1 (2ul)</td>
</tr>
<tr>
<td>g_stc_uart_config.parity</td>
<td>Parity bit</td>
<td>CY_SCB_UART_PARITY_NONE (0ul)</td>
</tr>
<tr>
<td>g_stc_uart_config.enableInputFilter</td>
<td>Median filter</td>
<td>False</td>
</tr>
<tr>
<td>g_stc_uart_config.dropOnParityError</td>
<td>Behavior when a parity check fails</td>
<td>False</td>
</tr>
<tr>
<td>g_stc_uart_config.dropOnFrameError</td>
<td>Behavior when an error is detected</td>
<td>False</td>
</tr>
<tr>
<td>g_stc_uart_config.multiProcess</td>
<td>Multi-processor mode</td>
<td>False (disable)</td>
</tr>
<tr>
<td>g_stc_uart_config.receiverAddress</td>
<td>Slave device address</td>
<td>0ul</td>
</tr>
<tr>
<td>g_stc_uart_config.receiverAddressMask</td>
<td>Slave device address mask</td>
<td>0ul</td>
</tr>
<tr>
<td>g_stc_uart_config.acceptAddrInFifo</td>
<td>Received matching address is accepted in the RX FIFO</td>
<td>False (not accept)</td>
</tr>
<tr>
<td>g_stc_uart_config.irdaInvertRx</td>
<td>Inverts incoming RX line signal</td>
<td>False</td>
</tr>
<tr>
<td>g_stc_uart_config.smartCardRetryOnAck</td>
<td>Data frame is retransmitted when a negative acknowledgement is received</td>
<td>False</td>
</tr>
<tr>
<td>g_stc_uart_config.enableCts</td>
<td>Enable the use of CTS input signal by the UART transmitter</td>
<td>False</td>
</tr>
<tr>
<td>g_stc_uart_config.ctsPolarity</td>
<td>Polarity of the CTS input signal</td>
<td>CY_SCB_UART_ACTIVE_LOW (0ul)</td>
</tr>
<tr>
<td>g_stc_uart_config.rtsRxFifoLevel</td>
<td>Trigger level</td>
<td>0ul</td>
</tr>
<tr>
<td>g_stc_uart_config.rtsPolarity</td>
<td>Polarity of the RTS output signal</td>
<td>CY_SCB_UART_ACTIVE_LOW (0ul)</td>
</tr>
<tr>
<td>g_stc_uart_config.breakWidth</td>
<td>Break width</td>
<td>0ul</td>
</tr>
<tr>
<td>g_stc_uart_config.rxFifoTriggerLevel</td>
<td>Trigger level for Rx FIFO</td>
<td>0ul</td>
</tr>
<tr>
<td>g_stc_uart_config.rxFifoIntEnableMask</td>
<td>Receiver interrupt mask</td>
<td>0ul</td>
</tr>
<tr>
<td>g_stc_uart_config.txFifoTriggerLevel</td>
<td>Trigger level for Tx FIFO</td>
<td>0ul</td>
</tr>
</tbody>
</table>

(table continues...)
4 UART setting procedure example

Table 6 (continued) List of UART mode configuration parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_stc_uart_config.txFifoIntEnableMask</td>
<td>Transmitter interrupt mask</td>
<td>0ul</td>
</tr>
<tr>
<td>CY_USB_SCB_TYPE</td>
<td>Using SCB channel number</td>
<td>SCB3</td>
</tr>
<tr>
<td>stc_port_pinCfg_uart.driveMode</td>
<td>GPIO drive mode</td>
<td>Rx: CY_GPIO_DM_HIGHZ (8ul)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tx: CY_GPIO_DM_STRONG_IN_OFF (6ul)</td>
</tr>
<tr>
<td>SPI_port_pin_cfg.hsiom</td>
<td>Specifies the connection for IO pin route</td>
<td>Rx: CY_USB_SCB_UART_RX_MUX (17ul)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tx: CY_USB_SCB_UART_TX_MUX (17ul)</td>
</tr>
<tr>
<td>CY_USB_SCB_UART_RX_PORT</td>
<td>GPIO port number for Rx</td>
<td>GPIO_PRT13</td>
</tr>
<tr>
<td>CY_USB_SCB_UART_RX_PIN</td>
<td>GPIO port pin for Rx</td>
<td>0ul</td>
</tr>
<tr>
<td>CY_USB_SCB_UART_TX_PORT</td>
<td>GPIO port number for Tx</td>
<td>GPIO_PRT13</td>
</tr>
<tr>
<td>CY_USB_SCB_UART_TX_PIN</td>
<td>GPIO port pin for Tx</td>
<td>1ul</td>
</tr>
</tbody>
</table>

Table 7 lists the UART functions of the driver part in SDL.

Table 7 List of functions

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cy_SCB_UART_GetNumInRxFifo (volatile stc_SCB_t const *base)</td>
<td>Returns the number of data elements in the UART Rx FIFO</td>
<td>*base: CY_USB_SCB_TYPE</td>
</tr>
<tr>
<td>Cy_SCB_UART_GetArray (volatile stc_SCB_t const *base, void *xBuf, uint32_t size)</td>
<td>Reads an array of data out of the UART Rx FIFO</td>
<td>*base: CY_USB_SCB_TYPE *xBuf: uart_in_data size: rx_num</td>
</tr>
<tr>
<td>Cy_SCB_UART_PutArray (volatile stc_SCB_t *base, void *xBuf, uint32_t size)</td>
<td>Places an array of data in the UART Tx FIFO</td>
<td>*base: CY_USB_SCB_TYPE *xBuf: uart_in_data size: rx_num</td>
</tr>
<tr>
<td>Cy_SCB_UART_DeInit (volatile stc_SCB_t *base)</td>
<td>De-initializes the SCB block</td>
<td>*base: CY_USB_SCB_TYPE</td>
</tr>
<tr>
<td>Cy_SCB_UART_Init (volatile stc_SCB_t *base, cy_stc_scb_uart_config_t const *config, cy_stc_scb_uart_context_t *context)</td>
<td>Initializes the SCB for UART operation</td>
<td>*base: CY_USB_SCB_TYPE  *config: g_stc_uart_config   *context: g_stc_uart_context</td>
</tr>
<tr>
<td>Cy_SCB_UART_Enable (volatile stc_SCB_t *base)</td>
<td>Enables the SCB block for the UART operation</td>
<td>*base: CY_USB_SCB_TYPE</td>
</tr>
</tbody>
</table>

Code Listing 19 demonstrates an example to configure UART mode in the configuration part.
4 UART setting procedure example

Code Listing 19 Example to configure UART mode in configuration part

```c
/* Select UART Echo Type */
/* Use Low-Level API. Polling & Receive by 1 byte unit */

/* Local Definition */
#define UART_OVERSAMPLING (8ul)

/* Local Functions Declaration */
void UART_Initialization(uint32_t boardrate, uint32_t sourceFreq);
void Term_Printf(void *fmt, ...);

*/ SCB - UART Configuration */
cy_stc_scb_uart_context_t g_stc_uart_context;
cy_stc_scb_uart_config_t g_stc_uart_config = /* Configure the UART parameters */ {
    .uartMode = CY_SCB_UART_STANDARD,
    .oversample = UART_OVERSAMPLING,
    .dataWidth = 8ul,
    .enableMsbFirst = false,
    .stopBits = CY_SCB_UART_STOP_BITS_1,
    .parity = CY_SCB_UART_PARITY_NONE,
    .enableInputFilter = false,
    .dropOnParityError = false,
    .dropOnFrameError = false,
    .enableMutiProcessorMode = false,
    .receiverAddress = 0ul,
    .receiverAddressMask = 0ul,
    .acceptAddrInFifo = false,
    .irdaInvertRx = false,
    .irdaEnableLowPowerReceiver = false,
    .smartCardRetryOnNack = false,
    .enableCts = false,
    .rtsPolarity = CY_SCB_UART_ACTIVE_LOW,
    .rtsRxFifoLevel = 0ul,
    .rtsPolarity = CY_SCB_UART_ACTIVE_LOW,
    .breakWidth = 0ul,
    .rxFifoTriggerLevel = 0ul,
    .rxFifoIntEnableMask = 0ul,
    .txFifoTriggerLevel = 0ul,
    .txFifoIntEnableMask = 0ul
};

int main(void)
{
    SystemInit();

    /* UART Initialization */
    //                  boardrate, peri frequency
    UART_Initialization( 115200ul, 80000000ul);
}
```
/* Opening UART Comment */
Term_Printf("\nUART TEST (driver ver=%d.%d)\n", CY_SCB_DRV_VERSION_MAJOR,
CY_SCB_DRV_VERSION_MINOR);
Term_Printf("POLLING 1BYTE ECHO\n");

for(;;)
{
    uint32_t    rx_num;
    /* Check Receive Data */
    rx_num = Cy_SCB_UART_GetNumInRxFifo(CY_USB_SCB_TYPE); /* Read the data from RX FIFO
    STATUS (See Code Listing 20) */
    if (rx_num != 0ul)
    {
        uint8_t uart_in_data[128];
        Cy_SCB_UART_GetArray(CY_USB_SCB_TYPE, uart_in_data, rx_num); /* (5) Read the data from RX FIFO (See Code Listing 21) */
        Cy_SCB_UART_PutArray(CY_USB_SCB_TYPE, uart_in_data, rx_num); /* (6) Write the data to TX FIFO (See Code Listing 22) */
    }
}

void UART_Initialization(uint32_t baudrate, uint32_t sourceFreq)
{
    /* Port Configuration for UART */
    cy_stc_gpio_pin_config_t    stc_port_pin_cfg_uart = {0ul}; /* (1) Set the UART port */
    stc_port_pin_cfg_uart.driveMode = CY_GPIO_DM_HIGHZ; /* (1) Set the UART port */
    stc_port_pin_cfg_uart.hsiom   = CY_USB_SCB_UART_RX_MUX; /* (1) Set the UART port */
    Cy_GPIO_Pin_Init(CY_USB_SCB_UART_RX_PORT, CY_USB_SCB_UART_RX_PIN, &stc_port_pin_cfg_uart); /* (1) Set the UART port */

    stc_port_pin_cfg_uart.driveMode = CY_GPIO_DM_STRONG_IN_OFF; /* (1) Set the UART port */
    stc_port_pin_cfg_uart.hsiom   = CY_USB_SCB_UART_TX_MUX; /* (1) Set the UART port */
    Cy_GPIO_Pin_Init(CY_USB_SCB_UART_TX_PORT, CY_USB_SCB_UART_TX_PIN, &stc_port_pin_cfg_uart); /* (1) Set the UART port */

    /* SCB-UART Initialization */
    Cy_SCB_UART_DeInit(CY_USB_SCB_TYPE); /* If necessary, stop the UART operation (See Code Listing 23) */
    Cy_SCB_UART_Init(CY_USB_SCB_TYPE, &g_stc_uart_config, &g_stc_uart_context); /* (2) Initialize SCB for UART (See Code Listing 24) */
    Cy_SCB_UART_Enable(CY_USB_SCB_TYPE); /* (3) Enable the UART (See Code Listing 25) */
How to use serial communications block (SCB) in TRAVEO™ T2G family

4 UART setting procedure example

```c
/* Clock Configuration for UART */
/* (4) Configure the clock */
// Assign a programmable divider
Cy_SysClk_PeriphAssignDivider(CY_USB_SCB_UART_PCLK, CY_SYSCLK_DIV_24_5_BIT, 0ul);
/* Configure the Peripheral Clock (See Code Listing 4) */
// Set divider value
{
    uint64_t targetFreq = UART_OVERSAMPLING * boardrate;
    uint64_t sourceFreq_fp5 = ((uint64_t)sourceFreq << 5ul);
    uint32_t divSetting_fp5 = (uint32_t)(sourceFreq_fp5 / targetFreq);
    Cy_SysClk_PeriphSetFracDivider(CY_SYSCLK_DIV_24_5_BIT, 0ul,
                                   (divSetting_fp5 & 0x1FFFFFE0ul) >> 5ul,
                                   (divSetting_fp5 & 0x0000001Ful));
}
// Enable peripheral divider
Cy_SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_24_5_BIT, 0ul); /* Enable the divider (See Code Listing 6) */
}

void Term_Printf(void *fmt, ...)
{
    uint8_t uart_out_data[128];
    va_list arg;

    /* UART Print */
    va_start(arg, fmt);
    vsprintf((char*)&uart_out_data[0], (char*)fmt, arg);
    while (Cy_SCB_UART_IsTxComplete(CY_USB_SCB_TYPE) != true) {};
    Cy_SCB_UART_PutArray(CY_USB_SCB_TYPE, uart_out_data, strlen((char*)uart_out_data));
    va_end(arg);
}
```

*1: For details, refer to the I/O System sections in the architecture TRM.

Code Listing 20 to Code Listing 25 demonstrate example program to configure the SCB in the driver part. The following description will help you understand the register notation of the driver part of the SDL:

**Code Listing 20 Cy_SCB_UART_GetNumInRxFifo() function**

```c
__STATIC_INLINE uint32_t Cy_SCB_UART_GetNumInRxFifo(volatile stc_SCB_t const *base)
{
    return Cy_SCB_GetNumInRxFifo(base); /* Read the data from RX FIFO STATUS */
}
```
Code Listing 21 Cy_SCB_UART_GetArray() function

```c
__STATIC_INLINE uint32_t Cy_SCB_UART_GetArray(const volatile stc_SCB_t *base, void *rxBuf, uint32_t size)
{
    return Cy_SCB_ReadArray(base, rxBuf, size); /* Read the data from RX FIFO */
}
```

Code Listing 22 Cy_SCB_UART_PutArray() function

```c
__STATIC_INLINE uint32_t Cy_SCB_UART_PutArray(const volatile stc_SCB_t *base, void *txBuf, uint32_t size)
{
    return Cy_SCB_WriteArray(base,txBuf, size); /* Write the data to TX FIFO */
}
```
void Cy_SCB_UART_DeInit(volatile stc_SCB_t *base)
{
    /* De-initialize the UART interface */ /* Set the default value to unCTRL and unUART_CTRL Reg */
    base->unCTRL.u32Register = CY_SCB_CTRL_DEF_VAL;
    base->unUART_CTRL.u32Register = CY_SCB_UART_CTRL_DEF_VAL;

    /* De-initialize the RX direction */ /* Set the default value to unUART_RX_CTRL, unRX_CTRL, unRX_FIFO_CTRL, and unRX_MATCH Reg */
    base->unUART_RX_CTRL.u32Register = 0ul;
    base->unRX_CTRL.u32Register = CY_SCB_RX_CTRL_DEF_VAL;
    base->unRX_FIFO_CTRL.u32Register = 0ul;
    base->unRX_MATCH.u32Register = 0ul;

    /* De-initialize the TX direction */ /* Set the default value to unUART_TX_CTRL, unTX_CTRL, and unTX_FIFO_CTRL Reg */
    base->unUART_TX_CTRL.u32Register = 0ul;
    base->unTX_CTRL.u32Register = CY_SCB_TX_CTRL_DEF_VAL;
    base->unTX_FIFO_CTRL.u32Register = 0ul;

    /* De-initialize the flow control */ /* Set the default value to unUART_FLOW_CTRL Reg */
    base->unUART_FLOW_CTRL.u32Register = 0ul;

    /* De-initialize the interrupt sources */ /* Set the default value to Reg for Interrupt */
    base->unINTR_SPI_EC_MASK.u32Register = 0ul;
    base->unINTR_I2C_EC_MASK.u32Register = 0ul;
    base->unINTR_RX_MASK.u32Register = 0ul;
    base->unINTR_TX_MASK.u32Register = 0ul;
    base->unINTR_M_MASK.u32Register = 0ul;
    base->unINTR_S_MASK.u32Register = 0ul;
}

How to use serial communications block (SCB) in TRAVEO™ T2G family

4 UART setting procedure example

Code Listing 23 Cy_SCB_UART_DeInit() function
Code Listing 24 Cy_SCB_UART_Init() function

```c
void Cy_SCB_UART_Init(void *base, cy_stc_scb_uart_config_t const *config, cy_stc_scb_uart_context_t *context)
{
    if ((NULL != base) && (NULL != config)) /* Check if configuration parameter values are valid */
    {
        uint32_t ovs;

        if (((CY_SCB_UART_IRDA == config->uartMode) && (!config->irdaEnableLowPowerReceiver))
            /* For Normal IrDA mode oversampling is always zero */
            ovs = 0ul;
        else
            ovs = (config->oversample - 1ul);

        /* Configure the UART interface */ /* Set the config value to unCTRL and unUART_CTRL Reg */
        temp_CTRL.stcField.u1ADDR_ACCEPT = (config->acceptAddrInFifo ? 1ul : 0ul);
        temp_CTRL.stcField.u2MEM_WIDTH = 0ul;
        temp_CTRL.stcField.u4OVS = ovs;
        temp_CTRL.stcField.u2MODE = CY_SCB_CTRL_MODE_UART;
        base->unCTRL.u32Register = temp_CTRL.u32Register;
        temp_UART_CTRL.stcField.u2MODE = config->uartMode;
        base->unUART_CTRL.u32Register = temp_UART_CTRL.u32Register;

        /* Configure the RX direction */ /* Set the config value to unUART_RX_CTRL, unRX_CTRL, and unRX_MATCH Reg */
        temp_UART_RX_CTRL.stcField.u1POLARITY = (config->irdaInvertRx ? 1ul : 0ul);
        temp_UART_RX_CTRL.stcField.u1MP_MODE = (config->enableMultiProcessorMode ? 1ul : 0ul);
        temp_UART_RX_CTRL.stcField.u1DROP_ON_PARITY_ERROR = (config->dropOnParityError ? 1ul : 0ul);
        temp_UART_RX_CTRL.stcField.u1DROP_ON_FRAME_ERROR = (config->dropOnFrameError ? 1ul : 0ul);
    }
}
```

How to use serial communications block (SCB) in TRAVEO™ T2G family

4 UART setting procedure example
How to use serial communications block (SCB) in TRAVEO™ T2G family

4 UART setting procedure example

```c
    temp_UART_RX_CTRL.stcField.u4BREAK_WIDTH = (config->breakWidth - 1ul);
    temp_UART_RX_CTRL.stcField.u3STOP_BITS = (config->stopBits - 1ul);
    temp_UART_RX_CTRL.stcField.u1PARITY = (config->parity & 0x00000001ul);
    temp_UART_RX_CTRL.stcField.u1PARITY_ENABLED = (config->parity & 0x00000002ul) >> 1;

    base->unUART_RX_CTRL.u32Register = temp_UART_RX_CTRL.u32Register;

    temp_RX_CTRL.stcField.u1MSB_FIRST = (config->enableMsbFirst ? 1ul : 0ul);
    temp_RX_CTRL.stcField.u1MEDIAN = (config->enableInputFilter ? 1ul : 0ul);
    temp_RX_CTRL.stcField.u5DATA_WIDTH = (config->dataWidth - 1ul);
    base->unRX_CTRL.u32Register = temp_RX_CTRL.u32Register;

    temp_RX_MATCH.stcField.u8ADDR = config->receiverAddress;
    temp_RX_MATCH.stcField.u8MASK = config->receiverAddressMask;
    base->unRX_MATCH.u32Register = temp_RX_MATCH.u32Register;

    /* Configure the TX direction */ /* Set the config value to unUART_TX_CTRL, unTX_CTRL, and unRX_FIFO_CTRL Reg */
    temp_UART_TX_CTRL.stcField.u1RETRY_ON_NACK = (config->smartCardRetryOnNack ? 1ul : 0ul);
    temp_UART_TX_CTRL.stcField.u3STOP_BITS = (config->stopBits - 1ul);
    temp_UART_TX_CTRL.stcField.u1PARITY = (config->parity & 0x00000001ul);
    temp_UART_TX_CTRL.stcField.u1PARITY_ENABLED = (config->parity & 0x00000002ul) >> 1;
    base->unUART_TX_CTRL.u32Register = temp_UART_TX_CTRL.u32Register;

    temp_TX_CTRL.stcField.u1MSB_FIRST = (config->enableMsbFirst ? 1ul : 0ul);
    temp_TX_CTRL.stcField.u5DATA_WIDTH = (config->dataWidth - 1ul);
    temp_TX_CTRL.stcField.u1OPEN_DRAIN = ((config->uartMode == CY_SCB_UART_SMARTCARD) ? 1ul : 0ul);
    base->unTX_CTRL.u32Register = temp_TX_CTRL.u32Register;

    temp_RX_FIFO_CTRL.stcField.u8TRIGGER_LEVEL = config->rxFifoTriggerLevel;
    base->unRX_FIFO_CTRL.u32Register = temp_RX_FIFO_CTRL.u32Register;

    /* Configure the flow control */ /* Set the config value to unUART_FLOW_CTRL, unTX_FIFO_CTRL Reg */
    temp_UART_FLOW_CTRL.stcField.u1CTS_ENABLED = (config->enableCts ? 1ul : 0ul);
    temp_UART_FLOW_CTRL.stcField.u1CTS_POLARITY = ((CY_SCB_UART_ACTIVE_HIGH == config->ctsPolarity) ? 1ul : 0ul);
    temp_UART_FLOW_CTRL.stcField.u1RTS_ENABLED = ((CY_SCB_UART_ACTIVE_HIGH == config->rtsPolarity) ? 1ul : 0ul);
    temp_UART_FLOW_CTRL.stcField.u8TRIGGER_LEVEL = config->txFifoTriggerLevel;
    base->unUART_FLOW_CTRL.u32Register = temp_UART_FLOW_CTRL.u32Register;

    temp_TX_FIFO_CTRL.stcField.u8TRIGGER_LEVEL = config->txFifoTriggerLevel;
    base->unTX_FIFO_CTRL.u32Register = temp_TX_FIFO_CTRL.u32Register;

    /* Set up interrupt sources */ /* Set the config value to unINTR_TX_MASK and
```
How to use serial communications block (SCB) in TRAVEO™ T2G family

4 UART setting procedure example

```c
unINTR_RX_MASK Reg */
    base->unINTR_TX_MASK.u32Register = config->txFifoIntEnableMask;
    base->unINTR_RX_MASK.u32Register = config->rxFifoIntEnableMask;

    /* Initialize context */
    if (NULL != context) {
        context->rxStatus = 0ul; /* Clear the context */
        context->txStatus = 0ul; /* Clear the context */

        context->rxRingBuf = NULL; /* Clear the context */
        context->rxRingBufSize = 0ul; /* Clear the context */

        context->txBufIdx = 0ul; /* Clear the context */
        context->txLeftToTransmit = 0ul; /* Clear the context */

        context->cbEvents = NULL; /* Clear the context */

        #if !defined(NDEBUG)
            /* Put an initialization key into the initKey variable to verify
             * context initialization in the transfer API.
             */
            context->initKey = CY_SCB_UART_INIT_KEY;
        #endif /* !(NDEBUG) */
    }
    retStatus = CY_SCB_UART_SUCCESS;

    return (retStatus);
}
```

Code Listing 25 Cy_SCB_UART_Enable() function

```c
__STATIC_INLINE void Cy_SCB_UART_Enable(volatile stc_SCB_t *base) {
    base->unCTRL.stcField.uiENABLED = 1ul; /* Set the enable bit to "1" */
}
```
5 I2C setting procedure example

This section shows an example of the SPI using the sample driver library (SDL). The SCB supports SPI Master mode and SPI Slave mode with Motorola, Texas Instruments, and National Semiconductor protocols. See the architecture TRM for details of each protocol. The code snippets in this application note are part of SDL. This sample program shows for CYT2B7 series. See Other references for the SDL.

This example shows the usage of the SCB in I2C mode. The SCB supports Master mode, Slave mode, and multi-Master mode. See the architecture TRM for details of each protocol.

5.1 Master mode

In this example, the SCB is configured as an I2C master and writes one byte data to the slave (address = 0x08) and reads one byte data from the same slave. For simplicity, polling method is used in this example instead of interrupts for writing and reading data to/from FIFOs.

5.1.1 Use case

- SCB mode = I2C Master mode
- SCB channel = 0
- PCLK = 2 MHz
- Bit rate = 100 kbps
  [Bit rate setting]

  The bit rate setting is valid only in Master mode. The bit rate calculation formula is as follows:

  Bit rate [bps] = Input clock [Hz] / (Low_phase_ovs + High_phase_ovs)

  Low_phase_ovs : SCB_I2C_CTRL.LOW_PHASE_OVS + 1
  High_phase_ovs : SCB_I2C_CTRL.HIGH_PHASE_OVS + 1

  In this case, bit rate is calculated as follows:

  Bit rate = Input clock [Hz] / (High_phase_ovs + Low_phase_ovs)
  = PCLK(2MHz) / ((9+1) + (9+1)) = 100 [kbps]

  For more details, see the architecture TRM.

- 7-bit Slave address = 0x8 (for another I2C device)
- MSb first
- Tx/Rx FIFO = Used
- Tx/Rx interrupt = Disabled
- Analog filter is enabled and digital filter = Disabled
- Used ports
  - SCL : SCB0_SCL (P1.0)
  - SDA : SCB0_SDA (P1.1)

Figure 8 shows the example of connection between the SCB and another I2C Slave device.
In I²C Master mode, SCL and SDA signals are connected to another I²C Slave device. The Master device outputs the clock (SCL) to the Slave device. The data signal (SDA) is bidirectional. Both SCL and SDA are pulled up to VDDIO via external pull up register.

**Figure 8** Example of I²C (Master mode) communication

*Note: (*1) For VDDIO value, see the datasheet (see Related documents).*

**Figure 9** shows setting procedure and operation example for I²C Master mode.
5  I2C setting procedure example

Figure 9  I2C Master mode operation

1. Configure the clock.
2. Configure the I2C port.
3. Initialize SCB for I2C.
4. Set the bit rate.
5. Enable I2C.
6. I2C communication for example
   a. Make sure Tx FIFO empty.
   b. Send START and receive ACK/NACK.
   c. Transmit 1-byte data.
   d. Send STOP.
5.1.2 Configuration and example

Configuration and example lists the parameters of the configuration part in SDL for I2C master mode.

Table 8 List of I2C Master mode configuration parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_SOURCE_CLK_FREQ</td>
<td>Frequency of input divider clock</td>
<td>800000000ul (80MHz)</td>
</tr>
<tr>
<td>E_I2C_INCLK_TARGET_FREQ</td>
<td>Frequency of peripheral clock</td>
<td>20000000ul (2MHz)</td>
</tr>
<tr>
<td>E_I2C_DATARATE</td>
<td>Baudrate for I2C</td>
<td>100000ul</td>
</tr>
<tr>
<td>USER_I2C_SCB_PCLK</td>
<td>Peripheral clock number</td>
<td>PCLK_SCB0_CLOCK</td>
</tr>
<tr>
<td>DIVIDER_NO_1</td>
<td>Divider number</td>
<td>1ul</td>
</tr>
<tr>
<td>E_I2C_SLAVE_ADDR</td>
<td>Slave device address</td>
<td>8ul</td>
</tr>
<tr>
<td>E_I2C_RECV_SIZE</td>
<td>Master buffer size</td>
<td>9ul</td>
</tr>
<tr>
<td>USER_I2C_SCB_TYPE</td>
<td>SCB channel number</td>
<td>SCB0</td>
</tr>
<tr>
<td>I2C_SDA_PORT</td>
<td>I/O port number</td>
<td>GPIO_PRT1</td>
</tr>
<tr>
<td>I2C_SDA_PORT_PIN</td>
<td>I/O pin number</td>
<td>1ul</td>
</tr>
<tr>
<td>I2C_SDA_PORT_MUX</td>
<td>Peripheral connection</td>
<td>P1_1_SCB0_I2C_SDA (14ul)</td>
</tr>
<tr>
<td>I2C_SCL_PORT</td>
<td>I/O port number</td>
<td>GPIO_PRT1</td>
</tr>
<tr>
<td>I2C_SCL_PORT_PIN</td>
<td>I/O pin number</td>
<td>0ul</td>
</tr>
<tr>
<td>I2C_SCL_PORT_MUX</td>
<td>Peripheral connection</td>
<td>P1_0_SCB0_I2C_SCL (14ul)</td>
</tr>
<tr>
<td>g_stc_i2c_config.i2cMode</td>
<td>I2C Master/Slave mode</td>
<td>CY_SCB_I2C_MASTER (2ul)</td>
</tr>
<tr>
<td>g_stc_i2c_config.useRxFifo</td>
<td>Receiver FIFO control</td>
<td>true</td>
</tr>
<tr>
<td>g_stc_i2c_config.useTxFifo</td>
<td>Transmitter FIFO control</td>
<td>true</td>
</tr>
<tr>
<td>g_stc_i2c_config.slaveAddress</td>
<td>Slave device address</td>
<td>E_I2C_SLAVE_ADDR (8ul)</td>
</tr>
<tr>
<td>g_stc_i2c_config.slaveAddressMask</td>
<td>Slave device address mask</td>
<td>E_I2C_SLAVE_ADDR (8ul)</td>
</tr>
<tr>
<td>g_stc_i2c_config.acceptAddrInFifo</td>
<td>Received matching address</td>
<td>false</td>
</tr>
<tr>
<td>g_stc_i2c_config.ackGeneralAddr</td>
<td>Received general call slave address</td>
<td>false</td>
</tr>
<tr>
<td>g_stc_i2c_config.enableWakeFromSleep</td>
<td>Clocking for the address matching</td>
<td>false</td>
</tr>
<tr>
<td>g_stc_i2c_master_config.slaveAddress</td>
<td>Slave device address</td>
<td>E_I2C_SLAVE_ADDR (8ul)</td>
</tr>
<tr>
<td>g_stc_i2c_master_config.buffer</td>
<td>Pointer to the master buffer</td>
<td>0ul</td>
</tr>
<tr>
<td>g_stc_i2c_master_config.bufferSize</td>
<td>Current master buffer size</td>
<td>0ul</td>
</tr>
<tr>
<td>g_stc_i2c_master_config.xferPending</td>
<td>Stores how the master ends the transfer</td>
<td>false</td>
</tr>
</tbody>
</table>
### Table 8 (continued) List of I2C Master mode configuration parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2S_port_pin_cfg.outVal</td>
<td>Pin output state</td>
<td>0ul</td>
</tr>
<tr>
<td>I2S_port_pin_cfg.driveMode</td>
<td>GPIO drive mode</td>
<td>0ul</td>
</tr>
<tr>
<td>I2S_port_pin_cfg.hsiom</td>
<td>Connection for I/O pin route</td>
<td>HSIOM_SEL_GPIO (0x0)</td>
</tr>
<tr>
<td>I2S_port_pin_cfg.intEdge</td>
<td>Edge which will trigger an IRQ</td>
<td>0ul</td>
</tr>
<tr>
<td>I2S_port_pin_cfg.intMask</td>
<td>Masks edge interrupt</td>
<td>0ul</td>
</tr>
<tr>
<td>I2S_port_pin_cfg.vtrip</td>
<td>Input buffer mode</td>
<td>0ul</td>
</tr>
<tr>
<td>I2S_port_pin_cfg.slewRate</td>
<td>Slew rate</td>
<td>0ul</td>
</tr>
<tr>
<td>I2S_port_pin_cfg.driveSel</td>
<td>GPIO drive strength</td>
<td>0ul</td>
</tr>
<tr>
<td>USER_I2C_SCB_IRQN</td>
<td>System interrupt index number</td>
<td>scb_0_interrupt_IRQn</td>
</tr>
<tr>
<td>irq_cfg.sysIntSrc</td>
<td>System interrupt index number</td>
<td>USER_I2C_SCB_IRQN (IDX: 17)</td>
</tr>
<tr>
<td>irq_cfg.intIdx</td>
<td>CPU interrupt number</td>
<td>CPUIntIdx3_IRQn</td>
</tr>
<tr>
<td>irq_cfg..isEnabled</td>
<td>CPU interrupt enable</td>
<td>true (enable)</td>
</tr>
</tbody>
</table>

Table 9 lists the I2C parameters of the driver part in the SDL.

### Table 9 List of functions

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>void Cy_SCB_I2C_DeInit</td>
<td>De-initializes the SCB block</td>
<td>*base: USER_I2C_SCB_TYPE</td>
</tr>
<tr>
<td>(volatile stc_SCB_t *base)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Cy_SCB_I2C_Init            | Initializes the SCB for the I2C operation | *base: USER_I2C_SCB_TYPE  
| (volatile stc_SCB_t *base,  |                              | *config: g_stc_i2c_config                  |
| cy_stc_scb_i2c_config_t const |                              | *context: g_stc_i2c_context                 |
| cy_stc_scb_i2c_context_t *context) |                              |                                              |
| Cy_SCB_I2C_SetDataRate     | Configures the SCB to work at the desired data rate | *base: USER_I2C_SCB_TYPE  
| (volatile stc_SCB_t *base,  |                              | dataRateHz: E_I2C_INCLK_TARGET_FREQ  
| uint32_t dataRateHz,       |                              | scbClockHz: E_I2C_INCLK_TARGET_FREQ       |
| uint32_t scbClockHz)       |                              |                                              |
| Cy_SCB_I2C_Enable          | Enables the SCB block for the I2C operation | *base: USER_I2C_SCB_TYPE                     |
| (volatile stc_SCB_t *base) |                              |                                              |
| Cy_SCB_GetNumInTxFifo      | Returns the number of data elements currently in the Tx FIFO | *base: USER_I2C_SCB_TYPE                     |
| (volatile stc_SCB_t const *base) |                              |                                              |
| Cy_SCB_I2C_MasterSendStart | Generates a start condition and sends a slave address with the Read/Write bit | *base: USER_I2C_SCB_TYPE  
| (volatile stc_SCB_t *base,  |                              | address: E_I2C_SLAVE_ADDR                  |
| uint32_t bitRnW,           |                              | bitRnW: CY_SCB_I2C_WRITE_XFER               |
| uint32_t timeoutMs,        |                              | timeoutMs: 2000ul                          |
| cy_stc_scb_i2c_context_t *context) |                              | *context: g_stc_i2c_context                 |
| (table continues...)       |                              |                                              |
### Table 9 (continued) List of functions

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>Remarks</th>
</tr>
</thead>
</table>
| Cy_SCB_I2C_MasterWriteByte (volatile stc_SCB_t *base, uint8_t theByte, uint32_t timeoutMs, cy_stc_scb_i2c_context_t *context) | Sends one byte to a slave | *base: USER_I2C_SCB_TYPE  
theByte: g_send_byte  
timeoutMs: 2000ul  
*context: g_stc_i2c_context) |
| Cy_SCB_I2C_MasterSendStop (volatile stc_SCB_t *base, uint32_t timeoutMs, cy_stc_scb_i2c_context_t *context) | Generates a Stop condition to complete the current transaction | *base: USER_I2C_SCB_TYPE  
timeoutMs: 2000ul  
*context: g_stc_i2c_context) |
| Cy_SCB_GetNumInRxFifo (volatile stc_SCB_t const *base) | Returns the number of data elements currently in the Rx FIFO | *base: USER_I2C_SCB_TYPE |
| Cy_SCB_I2C_MasterReadByte (volatile stc_SCB_t *base, guint32_t ackNack, uint8_t *byte, uint32_t timeoutMs, cy_stc_scb_i2c_context_t *context) | Reads one byte from a slave and generates an ACK or prepares to generate a NAK | *base: USER_I2C_SCB_TYPE  
ackNack: CY_SCB_I2C_NAK  
*byte: g_recv_byte  
timeoutMs: 2000ul  
*context: g_stc_i2c_context) |

**Code Listing 26** demonstrates an example to configure I²C master mode in the configuration part.
How to use serial communications block (SCB) in TRAVEO™ T2G family

5 I2C setting procedure example

Code Listing 26 Example to configure I2C master mode in configuration part

/* I2C Master Mode Test */
/* Partner Address(Slave): 0x08 (E_I2C_SLAVE_ADDR) */

#define USER_I2C_SCB_TYPE       SCB0 /* Define the SCB parameters */
#define USER_I2C_SCB_PCLK       PCLK_SCB0_CLOCK /* Define the SCB parameters */
#define USER_I2C_SCB_IRQN       scb_0_interrupt_IRQn /* Define the SCB parameters */
#define I2C_SDA_PORT     GPIO_PRT1 /* Define the port parameters */
#define I2C_SDA_PORT_PIN (1ul) /* Define the port parameters */
#define I2C_SDA_PORT_MUX P1_1_SCB0_I2C_SDA /* Define the port parameters */
#define I2C_SCL_PORT     GPIO_PRT1 /* Define the port parameters */
#define I2C_SCL_PORT_PIN (0ul) /* Define the port parameters */
#define I2C_SCL_PORT_MUX P1_0_SCB0_I2C_SCL /* Define the port parameters */
#define DIVIDER_NO_1 (1ul)

/* Select Frequency */
#define E_SOURCE_CLK_FREQ (80000000ul) /* Define the clock parameters */
#define E_I2C_INCLK_TARGET_FREQ (2000000ul) /* Define the clock parameters */
#define E_I2C_DATARATE (100000ul) /* Define the clock parameters */
#define E_I2C_SLAVE_ADDR 8ul
#define E_I2C_RECV_SIZE 9ul

static cy_stc_gpio_pin_config_t I2S_port_pin_cfg = /* Configure the port parameters */ {
    .outVal = 0ul,
    .driveMode = 0ul, /* Will be updated in runtime */
    .hsiom = HSIOM_SEL_GPIO, /* Will be updated in runtime */
    .intEdge = 0ul,
    .intMask = 0ul,
    .vtrip = 0ul,
    .slewRate = 0ul,
    .driveSel = 0ul,
};

/* SCB - I2C Configuration */
static cy_stc_scb_i2c_context_t g_stc_i2c_context;
static const cy_stc_scb_i2c_config_t  g_stc_i2c_config = {
    .i2cMode = CY_SCB_I2C_MASTER, /* Configure the I2C parameters */
    .useRxFifo = true, /* Configure the I2C parameters */
    .useTxFifo = true, /* Configure the I2C parameters */
    .slaveAddress = E_I2C_SLAVE_ADDR, /* Configure the I2C parameters */
    .slaveAddressMask = E_I2C_SLAVE_ADDR, /* Configure the I2C parameters */
    .acceptAddrInFifo = false, /* Configure the I2C parameters */
5 I²C setting procedure example

```c
void SetPeripheFracDiv24_5(uint64_t targetFreq, uint64_t sourceFreq, uint8_t divNum) /* Create the function to determine the divider division ratio */
{
    uint64_t temp = ((uint64_t)sourceFreq << 5ul);
    uint32_t divSetting;

    divSetting = (uint32_t)(temp / targetFreq); /* Calculate the division ratio */
    Cy_SysClk_PermSetFracDivider(CY_SYSCLK_DIV_24_5_BIT, divNum, /* Set the division ratio */
                                 (((divSetting >> 5ul) & 0x00000FFFul) - 1ul),
                                 (divSetting & 0x0000001Ful));
}

void Scb_I2C_Master_LowLevelAPI_Test(void)
{
    /*------------------------*/
    /* I2C Master Byte Write */
    /*------------------------*/
    /* Make sure TX FIFO empty */
    while(Cy_SCB_GetNumInTxFifo(USER_I2C_SCB_TYPE) != 0ul); /* (6-1) Make sure TX FIFO empty (See Code Listing 31) */

    /* Send START and Receive ACK/NACK */
    CY_ASSERT(Cy_SCB_I2C_MasterSendStart(USER_I2C_SCB_TYPE, E_I2C_SLAVE_ADDR, CY_SCB_I2C_WRITE_XFER, 2000ul, &g_stc_i2c_context) == CY_SCB_I2C_SUCCESS); /* (6-2) Send START and Receive ACK/NACK (See Code Listing 32) */

    /* Transmit One Byte Data */
    static uint8_t g_send_byte = 0xF1ul;
    CY_ASSERT(Cy_SCB_I2C_MasterWriteByte(USER_I2C_SCB_TYPE, g_send_byte, 2000ul, &g_stc_i2c_context) == CY_SCB_I2C_SUCCESS); /* (6-3) Transmit One Byte Data (See Code Listing 33) */

    /* Send STOP */
```
How to use serial communications block (SCB) in TRAEO™ T2G family

5  \textit{i}^2\text{C} setting procedure example

```c
CY_ASSERT(Cy_SCB_I2C_MasterSendWriteStop(USER_I2C_SCB_TYPE, 2000ul, &g_stc_i2c_context) == CY_SCB_I2C_SUCCESS); /* (6-4) Send STOP (See Code Listing 34) */

/*------------------------*/
/* I2C Master Byte Read   */
/*------------------------*/
/* Make sure RX FIFO empty */
while(Cy_SCB_GetNumInRxFifo(USER_I2C_SCB_TYPE) != 0ul); /* (6-5) Make sure RX FIFO empty (See Code Listing 35) */

/* Send START and Receive ACK/NACK */
CY_ASSERT(Cy_SCB_I2C_MasterSendStart(USER_I2C_SCB_TYPE, E_I2C_SLAVE_ADDR, CY_SCB_I2C_READ_XFER, 2000ul, &g_stc_i2c_context) == CY_SCB_I2C_SUCCESS); /* (6-6) Send START and Receive ACK/NACK (See Code Listing 32) */

/* Receive One Byte Data */
static uint8_t g_recv_byte = 0x00ul;
CY_ASSERT(Cy_SCB_I2C_Master.ReadByte(USER_I2C_SCB_TYPE, CY_SCB_I2C_NAK, &g_recv_byte, 2000ul, &g_stc_i2c_context) == CY_SCB_I2C_SUCCESS); /* (6-7) Receive One Byte Data (See Code Listing 36) */

/* Send NACK (and stop) */
CY_ASSERT(Cy_SCB_I2C_MasterSendReadStop(USER_I2C_SCB_TYPE, 2000ul, &g_stc_i2c_context) == CY_SCB_I2C_SUCCESS); /* (6-8) Send NACK and stop (See Code Listing 34) */
```

```c
int main(void)
{
    SystemInit();

    /*------------------------*/
    /* Clock Configuration */
    /*------------------------*/
    Cy_SysClk_PeriphAssignDivider(USER_I2C_SCB_PCLK, CY_SYSCLK_DIV_24_5_BIT, DIVIDER_NO_1); /* Configure the Peripheral Clock (See Code Listing 4) */
    SetPeripheFracDiv24_5(E_I2C_INCLK_TARGET_FREQ, E_SOURCE_CLK_FREQ, DIVIDER_NO_1); /* Configure the divider (See Code Listing 2) */
    Cy_SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_24_5_BIT, DIVIDER_NO_1); /* Enable the divider (See Code Listing 6) */

    /*------------------------*/
    /* Port Configuration */
```

Application note 56
/**------------------------*/
I2S_port_pin_cfg.driveMode = CY_GPIO_DM_OD_DRIVESLOW; /* (2) Configure the I2C port */
I2S_port_pin_cfg.hsiom = I2C_SDA_PORT_MUX; /* (2) Configure the I2C port */
Cy_GPIO_Pin_Init(I2C_SDA_PORT, I2C_SDA_PORT_PIN, &I2S_port_pin_cfg); /* (2) Configure the I2C port */

I2S_port_pin_cfg.driveMode = CY_GPIO_DM_OD_DRIVESLOW; /* (2) Configure the I2C port */
I2S_port_pin_cfg.hsiom = I2C_SCL_PORT_MUX; /* (2) Configure the I2C port */
Cy_GPIO_Pin_Init(I2C_SCL_PORT, I2C_SCL_PORT_PIN, &I2S_port_pin_cfg); /* (2) Configure the I2C port */

/**------------------------*/

/ * Initialize & Enable I2C */
/ *------------------------*/
Cy_SCB_I2C_DeInit(USER_I2C_SCB_TYPE); /* If necessary, stop the I2C operation (See Code Listing 27) */
Cy_SCB_I2C_Init(USER_I2C_SCB_TYPE, &g_stc_i2c_config, &g_stc_i2c_context); /* (3) Initialize SCB for I2C (See Code Listing 28) */
Cy_SCB_I2C_SetDataRate(USER_I2C_SCB_TYPE, E_I2C_DATARATE, E_I2C_INCLK_TARGET_FREQ); /* (4) Set the Bit Rate (See Code Listing 29) */
Cy_SCB_I2C_Enable(USER_I2C_SCB_TYPE); /* (5) Enable I2C (See Code Listing 30) */

/* I2C Master Mode Test */
Scb_I2C_Master_LowLevelAPI_Test(); /* (6) I2C Communication */

for(;;);

*/

*1: For details, refer to the I/O System sections in the architecture TRM.
Code Listing 27 to Code Listing 36 demonstrate example program to configure SCB in the driver part. The following description will help you understand the register notation of the driver part of the SDL:
void Cy_SCB_I2C_DeInit(volatile stc_SCB_t *base) {
    /* Returns block registers into the default state */
    base->unCTRL.u32Register = CY_SCB_CTRL_DEF_VAL; /* Set the default value to I2C_CTRL Reg */
    base->unI2C_CTRL.u32Register = CY_SCB_I2C_CTRL_DEF_VAL; /* Set the default value to I2C_CFG Reg */
    base->unI2C_CFG.u32Register = CY_SCB_I2C_CFG_DEF_VAL; /* Set the default value to I2C_CFG Reg */
    base->unRX_CTRL.u32Register = CY_SCB_RX_CTRL_DEF_VAL; /* Set the default value to RX related Reg */
    base->unRX_FIFO_CTRL.u32Register = 0ul; /* Set the default value to RX related Reg */
    base->unRX_MATCH.u32Register = 0ul; /* Set the default value to RX related Reg */
    base->unTX_CTRL.u32Register = CY_SCB_TX_CTRL_DEF_VAL; /* Set the default value to TX related Reg */
    base->unTX_FIFO_CTRL.u32Register = 0ul; /* Set the default value to TX related Reg */
    base->unINTR_SPI_EC_MASK.u32Register = 0ul; /* Set the default value to interrupt related Reg */
    base->unINTR_I2C_EC_MASK.u32Register = 0ul; /* Set the default value to interrupt related Reg */
    base->unINTR_RX_MASK.u32Register = 0ul; /* Set the default value to interrupt related Reg */
    base->unINTR_TX_MASK.u32Register = 0ul; /* Set the default value to interrupt related Reg */
    base->unINTR_M_MASK.u32Register = 0ul; /* Set the default value to interrupt related Reg */
    base->unINTR_S_MASK.u32Register = 0ul; /* Set the default value to interrupt related Reg */
}
Code Listing 28 Cy_SCB_I2C_Init() function

```c
// Function declaration

cy_en_scb_i2c_status_t Cy_SCB_I2C_Init(volatile stc_SCB_t *base, cy_stc_scb_i2c_config_t const *config, cy_stc_scb_i2c_context_t *context)
{
    cy_en_scb_i2c_status_t retStatus = CY_SCB_I2C_BAD_PARAM; /* Check if configuration parameter values are valid */
    un_SCB_CTRL_t         temp_CTRL = {0ul};
    un_SCB_I2C_CTRL_t     temp_I2C_CTRL = {0ul};
    un_SCB_RX_CTRL_t      temp_RX_CTRL = {0ul};
    un_SCB_RX_MATCH_t     temp_RX_MATCH = {0ul};
    un_SCB_TX_CTRL_t      temp_TX_CTRL = {0ul};

    if ((NULL != base) && (NULL != config) && (NULL != context))
    {
        /* Configure the I2C interface */ /* Init CTRL Reg */
        temp_CTRL.stcField.u1ADDR_ACCEPT = (config->acceptAddrInFifo ? 1ul : 0ul);
        temp_CTRL.stcField.u1EC_AM_MODE = (config->enableWakeFromSleep ? 1ul : 0ul);
        temp_CTRL.stcField.u2MEM_WIDTH = 0ul;
        temp_CTRL.stcField.u2MODE = CY_SCB_CTRL_MODE_I2C;
        base->unCTRL.u32Register = temp_CTRL.u32Register; /* Init CTRL */

        temp_I2C_CTRL.stcField.u1S_GENERAL_IGNORE = (config->ackGeneralAddr ? 0ul : 1ul);
        temp_I2C_CTRL.stcField.u1SLAVE_MODE = (config->i2cMode & 0x00000001ul);
        temp_I2C_CTRL.stcField.u1MASTER_MODE = (config->i2cMode & 0x00000002ul) >> 1ul;
        base->unI2C_CTRL.u32Register = temp_I2C_CTRL.u32Register; /* Init I2C_CTRL */

        /* Configure the RX direction */
        temp_RX_CTRL.stcField.u5DATA_WIDTH = CY_SCB_I2C_DATA_WIDTH;
        temp_RX_CTRL.stcField.u1MSB_FIRST = 1ul;
        base->unRX_CTRL.u32Register = temp_RX_CTRL.u32Register; /* Init RX_CTRL */

        temp_RX_MATCH.stcField.u8ADDR = ((uint32_t) config->slaveAddress << 1ul);
        temp_RX_MATCH.stcField.u8MASK = ((uint32_t) config->slaveAddressMask << 1ul);
        base->unRX_MATCH.u32Register = temp_RX_MATCH.u32Register; /* Init RX_MATCH */

        /* Configure the TX direction */
        temp_TX_CTRL.stcField.u5DATA_WIDTH = CY_SCB_I2C_DATA_WIDTH;
        temp_TX_CTRL.stcField.u1MSB_FIRST = 1ul;
        temp_TX_CTRL.stcField.u1OPEN_DRAIN = 1ul;
        base->unTX_CTRL.u32Register = temp_TX_CTRL.u32Register; /* Init TX_CTRL */

        base->unTX_FIFO_CTRL.u32Register = (config->useTxFifo ? CY_SCB_I2C_HALF_FIFO_SIZE : 1ul); /* Init TX_FIFO */

        /* Configure interrupt sources */ /* Init interrupt related Reg */
        base->unINTR_SPI_EC_MASK.u32Register = 0ul;
    }
}
```

How to use serial communications block (SCB) in TRAVEO™ T2G family

5 I²C setting procedure example
How to use serial communications block (SCB) in TRAVEO™ T2G family

5 I²C setting procedure example

```c
base->unINTR_I2C_EC_MASK.u32Register = 0ul;
base->unINTR_RX_MASK.u32Register = 0ul;
base->unINTR_TX_MASK.u32Register = 0ul;
base->unINTR_M_MASK.u32Register = 0ul;
base->unINTR_S_MASK.u32Register = ((0ul != (CY_SCB_I2C_SLAVE & config->i2cMode)) ? CY_SCB_I2C_SLAVE_INTR : 0ul));

    /* Initialize the context */
    context->useRxFifo = config->useRxFifo;
    context->useTxFifo = config->useTxFifo;

    context->state = CY_SCB_I2C_IDLE;

    /* Master-specific */
    context->masterStatus = 0ul;
    context->masterBufferIdx = 0ul;

    /* Slave-specific */
    context->slaveStatus = 0ul;
    context->slaveRxBufferIdx = 0ul;
    context->slaveRxBufferSize = 0ul;
    context->slaveTxBufferIdx = 0ul;
    context->slaveTxBufferSize = 0ul;

    /* Un-register callbacks */
    context->cbEvents = NULL;
    context->cbAddr = NULL;

    retStatus = CY_SCB_I2C_SUCCESS;
}

return (retStatus);
```
Code Listing 29 Cy_SCB_I2C_SetDataRate() function

```c
uint32_t Cy_SCB_I2C_SetDataRate(volatile stc_SCB_t *base, uint32_t dataRateHz, uint32_t scbClockHz)
{
    uint32_t actualDataRate = 0ul;

    if (((base->unI2C_CTRL.stcField.u1SLAVE_MODE == 1ul) && (base->unI2C_CTRL.stcField.u1MASTER_MODE == 0ul))
    {
        actualDataRate = Cy_SCB_I2C_GetDataRate(base, scbClockHz);
    /* Use an analog filter for the slave */
        base->unRX_CTRL.stcField.u1MEDIAN = 0ul;
        base->unI2C_CFG.u32Register = CY_SCB_I2C_ENABLE_ANALOG_FITLER;
    } else
    {
        if ((scbClockHz > 0u) && (dataRateHz > 0u))
        {
            uint32_t sclLow;
            uint32_t sclHigh;
            uint32_t lowPhase;
            uint32_t highPhase;

            /* Convert scb clock and data rate in kHz */
            uint32_t scbClockKHz = scbClockHz / 1000ul;
            uint32_t dataRateKHz = dataRateHz / 1000ul;

            /* Get period of the scb clock in ns */
            uint32_t period = 1000000000ul / scbClockHz;

            /* Get duration of SCL low and high for the selected data rate */
            if (dataRateHz <= CY_SCB_I2C_STD_DATA_RATE)
            {
                sclLow = CY_SCB_I2C_MASTER_STD_SCL_LOW;
                sclHigh = CY_SCB_I2C_MASTER_STD_SCL_HIGH;
            } else if (dataRateHz <= CY_SCB_I2C_FST_DATA_RATE)
            {
                sclLow = CY_SCB_I2C_MASTER_FST_SCL_LOW;
                sclHigh = CY_SCB_I2C_MASTER_FST_SCL_HIGH;
            } else
            {
                sclLow = CY_SCB_I2C_MASTER_FSTP_SCL_LOW;
                sclHigh = CY_SCB_I2C_MASTER_FSTP_SCL_HIGH;
            }

            /* Get low phase minimum value in scb clocks */
            lowPhase = sclLow / period;
            while (((period * lowPhase) < sclLow) && (lowPhase < CY_SCB_I2C_LOW_PHASE_MAX))
```
How to use serial communications block (SCB) in TRAVEO™ T2G family

5 i²C setting procedure example

```c
{
    ++lowPhase;
}

/* Get high phase minimum value in scb clocks */
highPhase = sclHigh / period;
while (((period * highPhase) < sclHigh) && (highPhase < CY_SCB_I2C_HIGH_PHASE_MAX))
{
    ++highPhase;
}

/* Get actual data rate */
actualDataRate = scbClockKHz / (lowPhase + highPhase);

uint32_t idx = 0ul;
while ((actualDataRate > dataRateKHz) &&
    ((lowPhase + highPhase) < CY_SCB_I2C_DUTY_CYCLE_MAX))
{
    /* Increase low and high phase to reach desired data rate */
    if (0ul != (idx & 0x1ul))
    {
        if (highPhase < CY_SCB_I2C_HIGH_PHASE_MAX)
        {
            highPhase++;
        }
    }
    else
    {
        if (lowPhase < CY_SCB_I2C_LOW_PHASE_MAX)
        {
            lowPhase++;
        }
    }
    idx++;

    /* Update actual data rate */
    actualDataRate = scbClockKHz / (lowPhase + highPhase);
}

/* Set filter configuration based on actual data rate */
if (actualDataRate > CY_SCB_I2C_FST_DATA_RATE)
{
    /* Use a digital filter */ /* Decide to use the Median filter */
    base->unRX_CTRL.stcField.u1MEDIAN = 1ul;
    base->unI2C_CFG.u32Register = CY_SCB_I2C_DISABLE_ANALOG_FITLER;
}
else
{
    /* Use an analog filter */
    base->unRX_CTRL.stcField.u1MEDIAN = 0ul;
    base->unI2C_CFG.u32Register = CY_SCB_I2C_ENABLE_ANALOG_FITLER;
}
```

Application note 62 002-25401 Rev. *E
2023-11-10
/* Set phase low and high */
Cy_SCB_I2C_MasterSetLowPhaseDutyCycle(base, lowPhase);
Cy_SCB_I2C_MasterSetHighPhaseDutyCycle(base, highPhase);

/* Convert actual data rate in Hz */
actualDataRate = scbClockHz / (lowPhase + highPhase); /* Calculate the I2C Bit Rate 
An example is shown below: 
Bit rate = Input Clock [Hz] / (High_phase_ovs + Low_phase_ovs) 
= PCLK(2MHz) / ((9+1) + (9+1)) = 100 [kbps] */ 

return (actualDataRate);

Code Listing 30 Cy_SCB_I2C_Enable() function

__STATIC_INLINE void Cy_SCB_I2C_Enable(volatile stc_SCB_t *base) 
{
    base->unCTRL.stcField.u1ENABLED = 1ul; /* Set the enable bit to “1” */
}

Code Listing 31 Cy_SCB_GetNumInTxFifo() function

__STATIC_INLINE uint32_t Cy_SCB_GetNumInTxFifo(volatile stc_SCB_t const *base) 
{
    return (base->unTX_FIFO_STATUS.stcField.u9USED); /* Make sure TX FIFO empty */
}
Code Listing 32 Cy_SCB_I2C_MasterSendStart() function

```c
void Cy_SCB_I2C_MasterSendStart(volatile stc_SCB_t *base, uint32_t address, uint32_t bitRnW, uint32_t timeoutMs, cy_stc_scb_i2c_context_t *context)
{
    cy_en_scb_i2c_status_t retStatus = CY_SCB_I2C_MASTER_NOT_READY;
    un_SCB_I2C_M_CMD_t temp_I2C_M_CMD;

    /* Disable the I2C slave interrupt sources to protect the state */
    Cy_SCB_SetSlaveInterruptMask(base, CY_SCB_CLEAR_ALL_INTR_SRC);

    if (CY_SCB_I2C_IDLE == context->state)
    {
        uint32_t locStatus;

        /* Convert the timeout to microseconds */ /* Start the timer for status check */
        uint32_t timeout = (timeoutMs * 1000UL);

        /* Set the read or write direction */
        context->state = CY_SCB_I2C_MASTER_ADDR;
        context->masterRdDir = (CY_SCB_I2C_READ_XFER == bitRnW);

        /* Clean up the hardware before a transfer. Note RX FIFO is empty at here */ /* Clear the status */
        Cy_SCB_ClearMasterInterrupt(base, CY_SCB_I2C_MASTER_INTR_ALL);
        Cy_SCB_ClearRxInterrupt(base, CY_SCB_RX_INTR_NOT_EMPTY);
        Cy_SCB_ClearTxFifo(base);

        /* Generate a Start and send address byte */
        Cy_SCB_WriteTxFifo(base, (_VAL2FLD(CY_SCB_I2C_ADDRESS, address) | bitRnW)); /* Send the address to slave */

        temp_I2C_M_CMD.u32Register = 0UL;
        temp_I2C_M_CMD.stcField.u1M_START_ON_IDLE = 1UL;
        base->unI2C_M_CMD.u32Register = temp_I2C_M_CMD.u32Register;

        /* Wait for a completion event from the master or slave */ /* Check the status */
        do
        {
            locStatus = ((CY_SCB_I2C_MASTER_TX_BYTE_DONE &
                          Cy_SCB_GetMasterInterruptStatus(base)) |
                          (CY_SCB_I2C_SLAVE_ADDR_DONE & Cy_SCB_GetSlaveInterruptStatus(base)));

            locStatus |= WaitOneUnit(&timeout);
        } while (0UL == locStatus);

        retStatus = HandleStatus(base, locStatus, context);
    }
}
```
How to use serial communications block (SCB) in TRAVEO™ T2G family

5  \( \text{i}^2\text{C} \) setting procedure example

```c
/* Enable I2C slave interrupt sources */
Cy_SCB_SetSlaveInterruptMask(base, CY_SCB_I2C_SLAVE_INTR);

return (retStatus);
}
```

**Code Listing 33 Cy_SCB_I2C_MasterWriteByte() function**

```c
cy_en_scb_i2c_status_t Cy_SCB_I2C_MasterWriteByte(volatile stc_SCB_t *base, uint8_t theByte,
                                                uint32_t timeoutMs,
                                                cy_stc_scb_i2c_context_t *context)
{
    cy_en_scb_i2c_status_t retStatus = CY_SCB_I2C_MASTER_NOT_READY;
    
    if (CY_SCB_I2C_MASTER_TX == context->state)
    {
        uint32_t locStatus;
        
        /* Convert the timeout to microseconds */ /* Start the timer for status check */
        uint32_t timeout = (timeoutMs * 1000ul);
        
        /* Send the data byte */
        Cy_SCB_WriteTxFifo(base, (uint32_t)theByte); /* Transmit One Byte Data */
        
        /* Wait for a completion event from the master or slave */ /* Check the status */
        do
        {
            locStatus = (CY_SCB_I2C_MASTER_TXBYTE_DONE &
                         Cy_SCB_GetMasterInterruptStatus(base));
            locStatus |= WaitOneUnit(&timeout);
        } while (0ul == locStatus);
        
        /* Convert the status from register plus timeout to the API status */
        retStatus = HandleStatus(base, locStatus, context);
    }
    
    return (retStatus);
}
```
Code Listing 34 Cy_SCB_I2C_MasterSendStop() function

```c
cy_en_scb_i2c_status_t Cy_SCB_I2C_MasterSendStop(volatile stc_SCB_t *base, uint32_t timeoutMs, cy_stc_scb_i2c_context_t *context)
{
    cy_en_scb_i2c_status_t retStatus = CY_SCB_I2C_MASTER_NOT_READY;
    un_SCB_I2C_M_CMD_t temp_I2C_M_CMD;

    if (0ul != (CY_SCB_I2C_MASTER_ACTIVE & context->state))
    {
        uint32_t locStatus;

        /* Convert the timeout to microseconds */
        uint32_t timeout = (timeoutMs * 1000ul);

        /* Generate a stop (for Write direction) and NACK plus stop for the Read direction */
        temp_I2C_M_CMD.u32Register = 0ul;
        temp_I2C_M_CMD.stcField.u1M_STOP = 1ul;
        temp_I2C_M_CMD.stcField.u1M_NACK = 1ul;
        base->unI2C_M_CMD.u32Register = temp_I2C_M_CMD.u32Register;

        /* Wait for a completion event from the master or slave */
        do
        {
            locStatus = (CY_SCB_I2C_MASTER_STOP_DONE & Cy_SCB_GetMasterInterruptStatus(base));

            locStatus |= WaitOneUnit(&timeout);
        } while (0ul == locStatus);

        /* Convert the status from register plus timeout to the API status */
        retStatus = HandleStatus(base, locStatus, context);
    }

    return (retStatus);
}
```

```c
cy_en_scb_i2c_status_t Cy_SCB_I2C_MasterSendWriteStop(volatile stc_SCB_t *base, uint32_t timeoutMs, cy_stc_scb_i2c_context_t *context) /*
Cy_SCB_I2C_MasterSendWriteStop */
{
    cy_en_scb_i2c_status_t retStatus = CY_SCB_I2C_MASTER_NOT_READY;

    if (0ul != (CY_SCB_I2C_MASTER_ACTIVE & context->state))
    {
        uint32_t locStatus;

        /* Convert the timeout to microseconds */ /* Start the timer for status check */
        uint32_t timeout = (timeoutMs * 1000ul);

        /* Generate a stop for Write direction */
```
base->unI2C_M_CMD.stcField.u1M_STOP = 1ul; /* Send to stop to slave device */

/* Wait for a completion event from the master or slave */ /* Check the status */
do {
    locStatus = (CY_SCB_I2C_MASTER_STOP_DONE & Cy_SCB_GetMasterInterruptStatus(base));
    locStatus |= WaitOneUnit(&timeout);
}
while (0ul == locStatus);

/* Convert the status from register plus timeout to the API status */
retStatus = HandleStatus(base, locStatus, context);

} return (retStatus);

cy_en_scb_i2c_status_t Cy_SCB_I2C_MasterSendReadStop(volatile stc_SCB_t *base, uint32_t timeoutMs, cy_stc_scb_i2c_context_t *context) {
    cy_en_scb_i2c_status_t retStatus = CY_SCB_I2C_MASTER_NOT_READY;
    if (0ul != (CY_SCB_I2C_MASTER_ACTIVE & context->state)) {
        uint32_t locStatus;

        /* Convert the timeout to microseconds */
        uint32_t timeout = (timeoutMs * 1000ul);

        /* Generate a NACK plus for the Read direction */
        base->unI2C_M_CMD.stcField.u1M_NACK = 1ul;

        /* Wait for a completion event from the master or slave */
        Do {
            locStatus = (CY_SCB_I2C_MASTER_STOP_DONE & Cy_SCB_GetMasterInterruptStatus(base));
            locStatus |= WaitOneUnit(&timeout);
        } while (0ul == locStatus);

        /* Convert the status from register plus timeout to the API status */
        retStatus = HandleStatus(base, locStatus, context);
    }

    return (retStatus);
}
How to use serial communications block (SCB) in TRAVEO™ T2G family

5 i2C setting procedure example

Code Listing 35 Cy_SCB_GetNumInRxFifo() function

```c
__STATIC_INLINE uint32_t Cy_SCB_GetNumInRxFifo(volatile stc_SCB_t const *base)
{
    return (base->unRX_FIFO_STATUS.stcField.u9USED); /* Make sure RX FIFO empty */
}
```
Code Listing 36 Cy_SCB_I2C_MasterReadByte() function

```c
cy_en_scb_i2c_status_t Cy_SCB_I2C_MasterReadByte(volatile stc_SCB_t *base, uint32_t ackNack,
                                                uint8_t *byte, uint32_t timeoutMs,
                                                cy_stc_scb_i2c_context_t *context)
{
    cy_en_scb_i2c_status_t retStatus = CY_SCB_I2C_MASTER_NOT_READY;
    un_SCB_I2C_M_CMD_t temp_I2C_M_CMD;

    if (CY_SCB_I2C_MASTER_RX0 == context->state)
        {
            bool rxEmpty;

            uint32_t locStatus;
            /* Convert the timeout to microseconds */
            uint32_t timeout = (timeoutMs * 1000ul); /* Start the timer for status check */

            /* Wait for ACK/NAK transmission and data byte reception */ /* Check the status */
            do
            {
                locStatus = (CY_SCB_I2C_MASTER_RX_BYTE_DONE &
                             Cy_SCB_GetMasterInterruptStatus(base));
                rxEmpty = (0ul == (CY_SCB_RX_INTR_NOT_EMPTY &
                                  Cy_SCB_GetRxInterruptStatus(base)));

                locStatus |= WaitOneUnit(&timeout);
            } while ((rxEmpty) && (0ul == locStatus));

            /* The Read byte if available */
            if (!rxEmpty)
            {
                /* Get the received data byte */
                *byte = (uint8_t) Cy_SCB_ReadRxFifo(base); /* Read the data */

                Cy_SCB_ClearRxInterrupt(base, CY_SCB_RX_INTR_NOT_EMPTY | CY_SCB_RX_INTR_LEVEL);
            }

            /* Convert the status from register plus timeout to the API status */
            retStatus = HandleStatus(base, locStatus, context);

            if (CY_SCB_I2C_SUCCESS == retStatus)
                {
                    /* Generate ACK or wait for NAK generation */
                    if (CY_SCB_I2C_ACK == ackNack)
                        {
                            temp_I2C_M_CMD.u32Register = 0ul;
                            temp_I2C_M_CMD.stcField.u1M_ACK = 1ul;
                            base->unI2C_M_CMD.u32Register = temp_I2C_M_CMD.u32Register;
                        }
                }
        }
```
5 I2C setting procedure example

```c
return (retStatus);
}
```

5.2 Slave mode

This example sets I2C Slave mode where the Master transmits the write or read data to the Slave SCB. If the Slave receives the data, an interrupt occurs, and the Slave decides whether it should perform the read or write procedure.

5.2.1 Use case

- SCB mode = I2C Slave mode
- SCB channel = 0
- PCLK = 2 MHz
- Bit rate = 100 kbps
- 7-bit Slave address = 0x8
- Tx/Rx FIFO = Used
- MSb first
- Data width = 8 bits
- Analog filter is enabled and digital filter = Disabled
- Enabled interrupts:
  - I2C_ARB_LOST (I2C Slave arbitration lost)
  - I2C_STOP (I2C STOP event detected)
  - I2C_ADDR_MATCH (I2C Slave address matching)
  - I2C_GENERAL (I2C Slave general call address received)
  - I2C_BUS_ERROR (I2C Slave bus error detected)
- Used ports
  - SCL : SCB0_SCL (P1.0)
  - SDA : SCB0_SDA (P1.1)

Figure 10 shows the example of the connection between the Slave SCB and another I2C Master device.

Note: (*)1 For VDDIO value, see the datasheet (see Related documents).
In I²C Slave mode, SCL and SDA signals are connected to another I²C Master device. The Master device outputs the clock (SCL) to the Slave device. The data (SDA) signal is bidirectional. Both SCL and SDA are pulled up to VDDIO via a resistor.

**Figure 11** shows the setting procedure and operation example for I²C Slave mode.

**5 I²C setting procedure example**

In I²C Slave mode, SCL and SDA signals are connected to another I²C Master device. The Master device outputs the clock (SCL) to the Slave device. The data (SDA) signal is bidirectional. Both SCL and SDA are pulled up to VDDIO via a resistor.

**Figure 11** shows the setting procedure and operation example for I²C Slave mode.

**How to use serial communications block (SCB) in TRAVEO™ T2G family**

5 I²C setting procedure example

In I²C Slave mode, SCL and SDA signals are connected to another I²C Master device. The Master device outputs the clock (SCL) to the Slave device. The data (SDA) signal is bidirectional. Both SCL and SDA are pulled up to VDDIO via a resistor.

**Figure 11** shows the setting procedure and operation example for I²C Slave mode.

**Figure 11**  I²C slave mode operation

1. Configure the clock.
2. Configure the I²C port.
3. Initialize the SCB for I²C.
4. Set the bit rate.
5. Configures the buffer pointer and size of the write buffer.
6. Configures the buffer pointer and the read buffer size.
7. Enable I²C.
8. Check the status.

a. Read CMPLT event: Configure the buffer pointer and the read buffer size
b. Write CMPLT event: Configure the buffer pointer and size of the write buffer
### Configuration and example

Table 10 lists the parameters of the configuration part in the SDL for I\(^2\)C slave mode.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_SOURCE_CLK_FREQ</td>
<td>Input divider clock frequency</td>
<td>80000000ul (80MHz)</td>
</tr>
<tr>
<td>E_I2C_INCLK_TARGET_FREQ</td>
<td>Peripheral clock frequency</td>
<td>20000000ul (2MHz)</td>
</tr>
<tr>
<td>E_I2C_DATARATE</td>
<td>I2C baud rate</td>
<td>1000000ul</td>
</tr>
<tr>
<td>USER_I2C_SCB_PCLK</td>
<td>Peripheral clock number</td>
<td>PCLK_SCB0_CLOCK</td>
</tr>
<tr>
<td>DIVIDER_NO_1</td>
<td>Divider number</td>
<td>1ul</td>
</tr>
<tr>
<td>E_I2C_SLAVE_ADDR</td>
<td>Slave device address</td>
<td>8ul</td>
</tr>
<tr>
<td>E_I2C_SLAVE_TXRX_BUF_SIZE</td>
<td>TXRX buffer size</td>
<td>32ul</td>
</tr>
<tr>
<td>E_I2C_SLAVE_USER_BUF_SIZE</td>
<td>User buffer size</td>
<td>32ul</td>
</tr>
<tr>
<td>USER_I2C_SCB_TYPE</td>
<td>SCB channel number</td>
<td>SCB0</td>
</tr>
<tr>
<td>I2C_SDA_PORT</td>
<td>I/O port number</td>
<td>GPIO_PRT1</td>
</tr>
<tr>
<td>I2C_SDA_PORT_PIN</td>
<td>I/O pin number</td>
<td>1ul</td>
</tr>
<tr>
<td>I2C_SDA_PORT_MUX</td>
<td>Peripheral connection</td>
<td>P1_1_SCB0_I2C_SDA (14ul)</td>
</tr>
<tr>
<td>I2C_SCL_PORT</td>
<td>I/O port number</td>
<td>GPIO_PRT1</td>
</tr>
<tr>
<td>I2C_SCL_PORT_PIN</td>
<td>I/O pin number</td>
<td>0ul</td>
</tr>
<tr>
<td>I2C_SCL_PORT_MUX</td>
<td>Peripheral connection</td>
<td>P1_0_SCB0_I2C_SCL (14ul)</td>
</tr>
<tr>
<td>g_stc_i2c_config.i2cMode</td>
<td>I2C Master/Slave mode</td>
<td>CY_SCB_I2C_SLAVE (1ul)</td>
</tr>
<tr>
<td>g_stc_i2c_config.useRx_fifo</td>
<td>Receiver FIFO control</td>
<td>true</td>
</tr>
<tr>
<td>g_stc_i2c_config.useTx_fifo</td>
<td>Transmitter FIFO control</td>
<td>true</td>
</tr>
<tr>
<td>g_stc_i2c_config.slaveAddress</td>
<td>Slave device address</td>
<td>E_I2C_SLAVE_ADDR (8ul)</td>
</tr>
<tr>
<td>g_stc_i2c_config.slaveAddressMask</td>
<td>Slave device address mask</td>
<td>0x7Ful</td>
</tr>
<tr>
<td>g_stc_i2c_config.acceptAddrInFifo</td>
<td>Received matching address</td>
<td>false</td>
</tr>
<tr>
<td>g_stc_i2c_config.ackGeneralAddr</td>
<td>Received general call slave address</td>
<td>true</td>
</tr>
<tr>
<td>g_stc_i2c_config.enableWakeFromSleep</td>
<td>Clocking for the address matching</td>
<td>false</td>
</tr>
<tr>
<td>I2S_port_pin_cfg.outVal</td>
<td>Pin output state</td>
<td>0ul</td>
</tr>
<tr>
<td>I2S_port_pin_cfg.driveMode</td>
<td>GPIO drive mode</td>
<td>0ul</td>
</tr>
<tr>
<td>I2S_port_pin_cfg.hsiom</td>
<td>Connection for I/O pin route</td>
<td>HSIOM_SEL_GPIO (0x0)</td>
</tr>
<tr>
<td>I2S_port_pin_cfg.intEdge</td>
<td>Edge which will trigger an IRQ</td>
<td>0ul</td>
</tr>
<tr>
<td>I2S_port_pin_cfg.intMask</td>
<td>Masks edge interrupt</td>
<td>0ul</td>
</tr>
<tr>
<td>I2S_port_pin_cfg.vtrip</td>
<td>Input buffer mode</td>
<td>0ul</td>
</tr>
<tr>
<td>I2S_port_pin_cfg.slewRate</td>
<td>Slew rate</td>
<td>0ul</td>
</tr>
</tbody>
</table>

*(table continues...)*
### Table 10 (continued) List of I²C slave mode configuration parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Setting value</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2S_port_pin_cfg.driveSel</td>
<td>GPIO drive strength</td>
<td>0ul</td>
</tr>
<tr>
<td>USER_I2C_SCB_IRQN</td>
<td>System interrupt index number</td>
<td>scb_0_interrupt_IRQn</td>
</tr>
<tr>
<td>irq_cfg.sysIntSrc</td>
<td>System interrupt index number</td>
<td>USER_I2C_SCB_IRQN (IDX: 17)</td>
</tr>
<tr>
<td>irq_cfg.intIdx</td>
<td>CPU interrupt number</td>
<td>CPUIntIdx3_IRQn</td>
</tr>
<tr>
<td>irq_cfg.isEnabled</td>
<td>CPU interrupt enable</td>
<td>true (enable)</td>
</tr>
</tbody>
</table>

Table 11 lists the I²C parameters of the driver part in the SDL.

### Table 11 List of functions

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cy_SCB_I2C_SlaveConfigWriteBuf</td>
<td>Configures the buffer pointer and size of the write buffer</td>
<td>*base: USER_I2C_SCB_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*wrBuf: g_i2c_rx_buf[0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>size: E_I2C_SLAVE_TXRX_BUF_SIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*context: g_stc_i2c_context</td>
</tr>
<tr>
<td>Cy_SCB_I2C_RegisterEventCallback</td>
<td>Registers a callback function that notifies</td>
<td>*base: USER_I2C_SCB_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>callback: Scb_I2C_Slave_Event</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*context: g_stc_i2c_context</td>
</tr>
<tr>
<td>Cy_SCB_I2C_SlaveConfigReadBuf</td>
<td>Configures the buffer pointer and the read buffer size</td>
<td>*base: USER_I2C_SCB_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*rdBuf: g_i2c_rx_buf[0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>size: E_I2C_SLAVE_TXRX_BUF_SIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*context: g_stc_i2c_context</td>
</tr>
</tbody>
</table>

Code Listing 37 demonstrates an example to configure I²C master mode in the configuration part.
Code Listing 37 Example to configure I^2C slave mode in configuration part

```c
/* I^2C Slave Mode Test */
/* Partner Address(Slave): 0x08 (E_I2C_SLAVE_ADDR) */

#define USER_I2C_SCB_TYPE SCB0 /* Define the SCB parameters */
#define USER_I2C_SCB_PCLK PCLK_SCB0_CLOCK /* Define the SCB parameters */
#define USER_I2C_SCB_IRQN scb_0_interrupt_IRQn /* Define the SCB parameters */

#define I2C_SDA_PORT GPIO_PRT1 /* Define the port parameters */
#define I2C_SDA_PORT_PIN (1ul) /* Define the port parameters */
#define I2C_SDA_PORT_MUX P1_1_SCB0_I2C_SDA /* Define the port parameters */

#define I2C_SCL_PORT GPIO_PRT1 /* Define the port parameters */
#define I2C_SCL_PORT_PIN (0ul) /* Define the port parameters */
#define I2C_SCL_PORT_MUX P1_0_SCB0_I2C_SCL /* Define the port parameters */

#define DIVIDER_NO_1 (1ul)

/* Select Frequency */
#define E_SOURCE_CLK_FREQ (80000000ul) /* Define the clock parameters */
#define E_I2C_INCLK_TARGET_FREQ (2000000ul) /* Define the clock parameters */
#define E_I2C_DATARATE (100000ul) /* Define the clock parameters */

#define E_I2C_SLAVE_ADDR 8ul
#define E_I2C_SLAVE_TXRX_BUF_SIZE 32ul
#define E_I2C_SLAVE_USER_BUF_SIZE 32ul // should be 2^n

static cy_stc_gpio_pin_config_t I2S_port_pin_cfg = /* Configure the port parameters */{
    .outVal = 0ul,
    .driveMode = 0ul, /* Will be updated in runtime */
    .hsiom = HSIOM_SEL_GPIO, /* Will be updated in runtime */
    .intEdge = 0ul,
    .intMask = 0ul,
    .vtrip = 0ul,
    . slewRate = 0ul,
    .driveSel = 0ul,
};

/* SCB - I^2C Configuration */
static cy_stc_scb_i2c_context_t    g_stc_i2c_context; /* Configure the I^2C parameters */
static cy_stc_scb_i2c_config_t     g_stc_i2c_config = /* Configure the I^2C parameters */{
    .i2cMode = CY_SCB_I2C_SLAVE, /* Configure the I^2C parameters */
    .useRxFifo = true, /* Configure the I^2C parameters */
    .useTxFifo = true, /* Configure the I^2C parameters */
    .slaveAddress = E_I2C_SLAVE_ADDR, /* Configure the I^2C parameters */
    .slaveAddressMask = 0x7Ful, /* Configure the I^2C parameters */
    .acceptAddrInFifo = false, /* Configure the I^2C parameters */
    .ackGeneralAddr = true, /* Configure the I^2C parameters */
};
```

How to use serial communications block (SCB) in TRAVEO™ T2G family

5 I^2C setting procedure example

Application note 74 002-25401 Rev. *E 2023-11-10
.enableWakeFromSleep = false /* Configure the I2C parameters */

/* Local Variables */
static uint8_t g_i2c_tx_buf[E_I2C_SLAVE_TXRX_BUF_SIZE] =
{ 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x50,
  0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x60,
  0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x70,
  0x71, 0x72};
static uint8_t          g_i2c_rx_buf[E_I2C_SLAVE_TXRX_BUF_SIZE];
static volatile uint8_t g_i2c_user_buf[E_I2C_SLAVE_USER_BUF_SIZE] = {0ul};
static uint8_t          g_i2c_user_buf_index = 0ul;

void SetPeripherFracDiv24_5(uint64_t targetFreq, uint64_t sourceFreq, uint8_t divNum)
{
  uint64_t temp = ((uint64_t)sourceFreq << 5ul);
  uint32_t divSetting;
  divSetting = (uint32_t)(temp / targetFreq);
  Cy_SysClk_PeriphSetFracDivider(CY_SYSCLK_DIV_24_5_BIT, divNum,
  (((divSetting >> 5ul) & 0x00000FFFul) - 1ul),
  (divSetting & 0x0000001Ful));
}

void Scb_I2C_Slave_Event(uint32_t locEvents)
{
  uint32_t recv_size;
  switch (locEvents) /* (8) Check the status */
  {
    case CY_SCB_I2C_SLAVE_READ_EVENT:
      break;
    case CY_SCB_I2C_SLAVE_WRITE_EVENT:
      break;
    case CY_SCB_I2C_SLAVE_RD_IN_FIFO_EVENT:
      break;
    case CY_SCB_I2C_SLAVE_RD_BUF_EMPTY_EVENT:
      break;
    case CY_SCB_I2C_SLAVE_RD_CMPLT_EVENT: /* (8-1) Read CMPLT Event */
      /* Clear Read Buffer (use same buffer) */
      Cy_Scb_I2C_SlaveConfigReadBuf(USER_I2C_SCB_TYPE, &g_i2c_tx_buf[0],
        E_I2C_SLAVE_TXRX_BUF_SIZE, &g_stc_i2c_context /* Configures the buffer pointer and the read buffer size (See Code Listing 40) */);
      break;
    case CY_SCB_I2C_SLAVE_WR_CMPLT_EVENT: /* (8-2) Write CMPLT Event */
      /* Copy Received Data to User Buffer(g_i2c_user_buf[32]) */
      recv_size = Cy_Scb_I2C_SlaveGetWriteTransferCount(USER_I2C_SCB_TYPE, &g_stc_i2c_context);
      for(uint32_t i = 0ul; i < recv_size; i++)
        {
How to use serial communications block (SCB) in TRAVEO™ T2G family

5 I2C setting procedure example

```c
    g_i2c_user_buf[g_i2c_user_buf_index] = g_i2c_rx_buf[1];
    g_i2c_user_buf_index = (g_i2c_user_buf_index + 1ul) & (E_I2C_SLAVE_USER_BUF_SIZE -
1ul);
}
/* Clear Write Buffer */
    Cy_SCB_I2C_SlaveConfigWriteBuf(USER_I2C_SCB_TYPE, &g_i2c_rx_buf[0],
E_I2C_SLAVE_TXRX_BUF_SIZE, &g_stc_i2c_context /* Configures the buffer pointer and size of the
write buffer (See Code Listing 38) */;
    break;
    case CY_SCB_I2C_SLAVE_ERR_EVENT:
        break;
    default:
        break;
}
```

```c
int main(void)
{
    SystemInit();

    /*---------------------*/
    /* Clock Configuration */ /* (1) Configure the clock */
    /*---------------------*/
    Cy_SysClk_PeriphAssignDivider(USER_I2C_SCB_PCLK, CY_SYSCLK_DIV_24_5_BIT,
DIVIDER_NO_1); /* Configure the Peripheral Clock (See Code Listing 4) */
    SetPeripheFracDiv24_5(E_I2C_INCLK_TARGET_FREQ, E_SOURCE_CLK_FREQ, DIVIDER_NO_1); /*
Configure the driver (See Code Listing 2) */
    Cy_SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_24_5_BIT, DIVIDER_NO_1); /* Enable the divider
(See Code Listing 5) */

    /*---------------------*/
    /* Port Configuration */
    /*---------------------*/
    I2S_port_pin_cfg.driveMode = CY_GPIO_DM_OD_DRIVESLOW; /* Configure the I2C port */
    I2S_port_pin_cfg.hsiom = I2C_SDA_PORT_MUX; /* Configure the I2C port */
    Cy_GPIO_Pin_Init(I2C_SDA_PORT, I2C_SDA_PORT_PIN, &I2S_port_pin_cfg); /* Configure the I2C
port */

    I2S_port_pin_cfg.driveMode = CY_GPIO_DM_OD_DRIVESLOW; /* Configure the I2C port */
    I2S_port_pin_cfg.hsiom = I2C_SCL_PORT_MUX; /* Configure the I2C port */
    Cy_GPIO_Pin_Init(I2C_SCL_PORT, I2C_SCL_PORT_PIN, &I2S_port_pin_cfg); /* Configure the I2C
port */
```

Application note 76 002-25401 Rev. *E
2023-11-10
How to use serial communications block (SCB) in TRAVEO™ T2G family

5 I²C setting procedure example

```c
/*--------------------------*/
/* Initialize & Enable I2C */
/*--------------------------*/
Cy_SCB_I2C_DeInit(USER_I2C_SCB_TYPE);  /* If necessary, stop the I2C operation (See Code Listing 27) */
Cy_SCB_I2C_Init(USER_I2C_SCB_TYPE, &g_stc_i2c_config, &g_stc_i2c_context);  /* (3) Initialize SCB for I2C (See Code Listing 28) */

Cy_SCB_I2C_SetDataRate(USER_I2C_SCB_TYPE, E_I2C_DATARATE, E_I2C_INCLK_TARGET_FREQ);  /* (4) Set the Bit Rate (See Code Listing 29) */

Cy_SCB_I2C_SlaveConfigWriteBuf(USER_I2C_SCB_TYPE, &g_i2c_rx_buf[0], E_I2C_SLAVE_TXRX_BUF_SIZE, &g_stc_i2c_context);  /* (5) Configures the buffer pointer and size of the write buffer (See Code Listing 38) */

Cy_SCB_I2C_SlaveConfigReadBuf(USER_I2C_SCB_TYPE, &g_i2c_tx_buf[0], E_I2C_SLAVE_TXRX_BUF_SIZE, &g_stc_i2c_context);  /* (6) Configures the buffer pointer and the read buffer size (See Code Listing 40) */

Cy_SCB_I2C_RegisterEventCallback(USER_I2C_SCB_TYPE, scb_i2c_handle_events_t Scb_I2C_Slave_Event, &g_stc_i2c_context);  /* Jump the (8) (See Code Listing 39) */

Cy_SCB_I2C_Enable(USER_I2C_SCB_TYPE);  /* (7) Enable I2C (See Code Listing 30) */

for(;;);
```

*1: For details, see the I/O System sections in the architecture TRM.

Code Listing 38 to Code Listing 40 demonstrate example program to configure the SCB in the driver part. The following description will help you understand the register notation of the driver part of the SDL:
How to use serial communications block (SCB) in TRAVEO™ T2G family

5 I²C setting procedure example

Code Listing 38 CY_SCB_I2C_SlaveConfigWriteBuf() function

```c
void Cy_SCB_I2C_SlaveConfigWriteBuf(volatile stc_SCB_t const *base, uint8_t *wrBuf, uint32_t size, cy_stc_scb_i2c_context_t *context)
{
    /* Suppress a compiler warning about unused variables */
    (void) base;
    CY_ASSERT(((NULL == wrBuf) && (0u == size)) || (NULL != wrBuf)); /* Write the data to buffer */
    context->slaveRxBuffer = wrBuf;
    context->slaveRxBufferSize = size;
    context->slaveRxBufferIdx = 0ul;
}
```

Code Listing 39 Cy_SCB_I2C_RegisterEventCallback() function

```c
__STATIC_INLINE void Cy_SCB_I2C_RegisterEventCallback(volatile stc_SCB_t const *base, scb_i2c_handle_events_t callback, cy_stc_scb_i2c_context_t *context)
{
    /* Suppress a compiler warning about unused variables */
    (void) base;
    context->cbEvents = callback; /* Function (Scb_I2C_Slave_Event) callback */
}
```

Code Listing 40 CY_SCB_I2C_SlaveConfigReadBuf() function

```c
void Cy_SCB_I2C_SlaveConfigReadBuf(volatile stc_SCB_t const *base, uint8_t *rdBuf, uint32_t size, cy_stc_scb_i2c_context_t *context)
{
    /* Suppress a compiler warning about unused variables */
    (void) base;
    CY_ASSERT(((NULL == rdBuf) && (0u == size)) || (NULL != rdBuf)); /* Read the data from buffer */
    context->slaveTxBuffer = rdBuf;
    context->slaveTxBufferSize = size;
    context->slaveTxBufferIdx = 0ul;
    context->slaveTxBufferCnt = 0ul;
}
```
## Glossary

### Table 12 Glossary

<table>
<thead>
<tr>
<th>Terms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD_RESP mode</td>
<td>CMD_RESP (command response) mode is similar to EZ mode. The major difference is whether a CPU sets the Slave's base address or a Master device sets it.</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct memory access</td>
</tr>
<tr>
<td>EZ mode</td>
<td>EZ (easy) mode is the Infineon original communication protocol which is prepared to simplify the Write/Read access between the device in SPI and I2C. During DeepSleep mode, it can communicate with the Master device without CPU intervention.</td>
</tr>
<tr>
<td>FIFO</td>
<td>First in First Out</td>
</tr>
<tr>
<td>I2C</td>
<td>Inter-Integrated Circuit</td>
</tr>
<tr>
<td></td>
<td>I2C bus is a serial synchronous communication bus corresponding to the multi-Master and the multi-Slave. It is used for low-speed communication between MCUs and peripheral devices. I2C bus is used with two lines of clock (SCK) and data (SDA), and usually pulled-up by resistance.</td>
</tr>
<tr>
<td>IrDA</td>
<td>IrDA is a kind of the standards of the optical radio data communication by infrared rays.</td>
</tr>
<tr>
<td>LIN</td>
<td>Local Interconnect Network</td>
</tr>
<tr>
<td></td>
<td>LIN is a serial communication network for automotive. It is used for the data communication between a control unit and various sensors/actuators. LIN takes lower cost than CAN.</td>
</tr>
<tr>
<td>Smart card</td>
<td>Smart card is a card which integrated a circuit to record data and to operate it.</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial peripheral interface</td>
</tr>
<tr>
<td></td>
<td>SPI is a synchronous serial communication interface specification used for short distance communication with peripheral devices.</td>
</tr>
<tr>
<td>UART</td>
<td>Universal asynchronous receiver-transmitter</td>
</tr>
<tr>
<td></td>
<td>UART is a receiver-transmitter circuit to convert a serial signal into a parallel signal, and to convert the opposite direction. It is used for low-speed communication between MCU and an external equipment.</td>
</tr>
</tbody>
</table>
7 Related documents

The following are the TRAVEO™ T2G family series datasheets and technical reference manuals. Contact Technical Support to obtain these documents.

• Device datasheet
  - CYT2B7 datasheet 32-Bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
  - CYT2B9 datasheet 32-Bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
  - CYT4BF datasheet 32-Bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
  - CYT4DN datasheet 32-Bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-24601)
  - CYT3BB/4BB datasheet 32-Bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-27763)

• Body controller entry family
  - TRAVEO™ T2G automotive body controller entry family architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B7
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B9

• Body controller high family
  - TRAVEO™ T2G automotive body controller high family architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT4BF
  - TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT3BB/4BB

• Cluster 2D family
  - TRAVEO™ T2G automotive cluster 2D family architecture technical reference manual (TRM) (Doc No. 002-25800)
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT4DN (Doc No. 002-25923)
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT3DL (Doc No. 002-29854)
8 Other references

Infineon provides the sample driver library (SDL) including startup code as sample software to access various peripherals. SDL also serves as a reference to customers for drivers that are not covered by the official AUTOSAR products. The SDL cannot be used for production purposes because it does not qualify to any automotive standards. The code snippets in this application note are part of the SDL. Contact Technical Support to obtain the SDL.
Revision history

<table>
<thead>
<tr>
<th>Document version</th>
<th>Date of release</th>
<th>Description of changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>2019-07-09</td>
<td>New application note.</td>
</tr>
<tr>
<td>*C</td>
<td>2021-06-08</td>
<td>Updated to Infineon template. Completing Sunset Review.</td>
</tr>
<tr>
<td>*D</td>
<td>2022-03-08</td>
<td>Updated code examples using SDL</td>
</tr>
<tr>
<td>*E</td>
<td>2023-11-10</td>
<td>Template update; no content update</td>
</tr>
</tbody>
</table>