Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

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

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

AN225401

How to Use Serial Communications Block (SCB) in Traveo II Family

Author: Koichi Tsuchiya

Associated Part Family: Traveo™ II Family CYT2/CYT3/CYT4 Series

Related Application Notes: see Related Documents.

AN225401 demonstrates how to configure and use a Serial Communications Block (SCB) in Traveo™ II family MCU with three serial interface protocols: SPI, UART, and I²C.

Contents

1 Introduction

This application note describes how to use a Serial Communications Block (SCB) in Cypress Traveo II family 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 (Architecture 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 II 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.

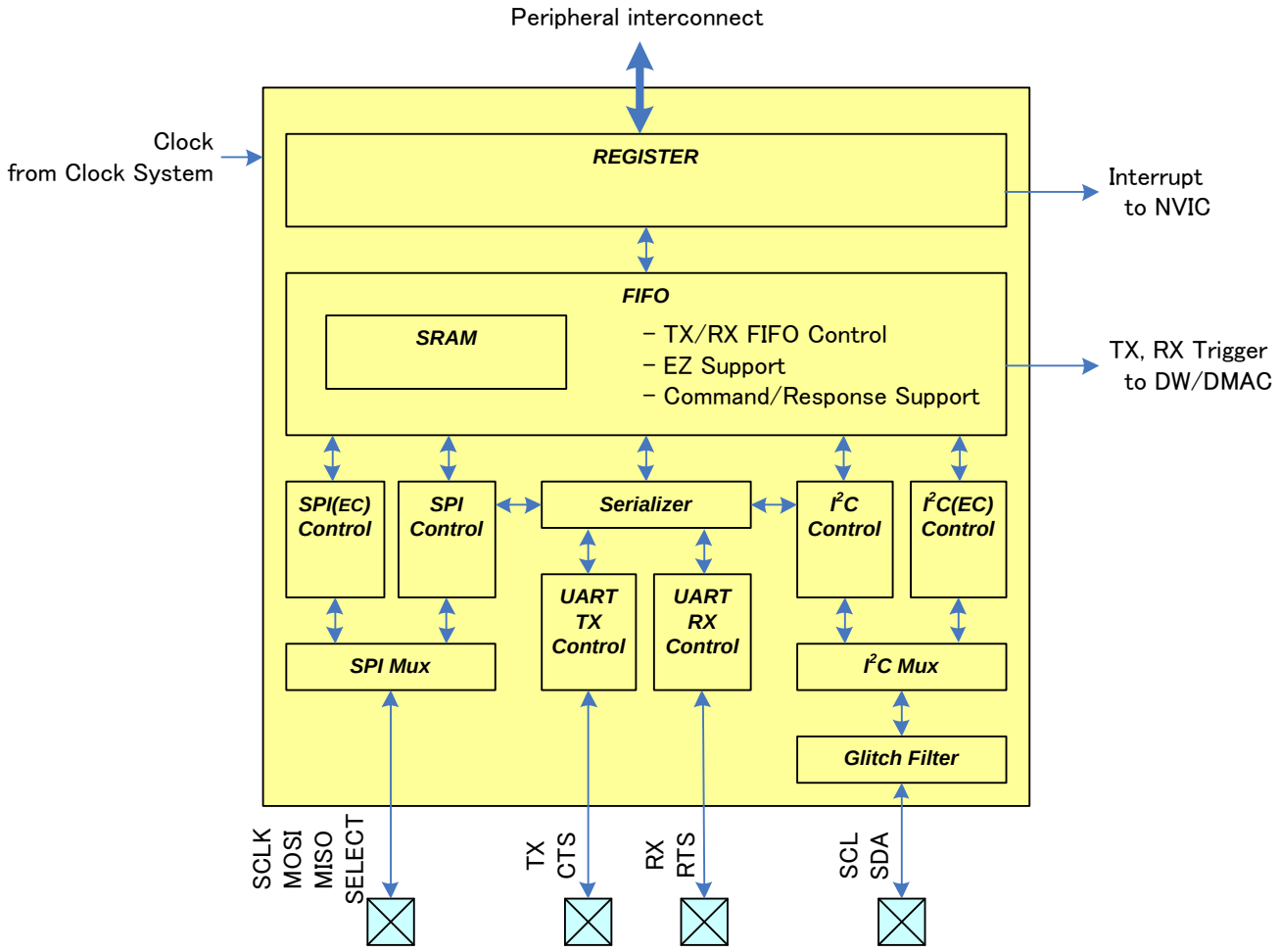
2 General Description

The SCB supports three serial communication protocols: SPI, UART, and I²C. Only one of the protocol 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 TRM.

Figure 1 shows the block diagram of SCB.

Figure 1. Block Diagram of SCB

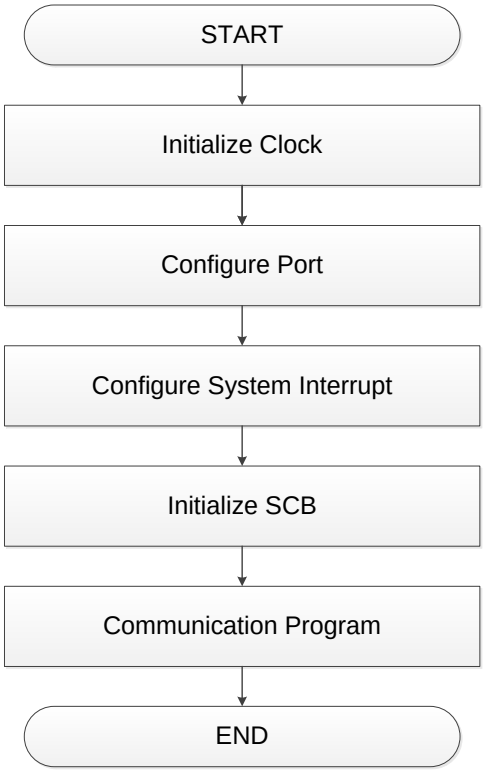


The SCB consists of registers, FIFO, and a control block for each protocol function (SPI, UART, and I²C). Registers are used as a software interfaces for SCB settings and generated interrupts by each event. The FIFO consists of SRAM (256-byte) and has three modes Tx/Rx FIFO (128x8-bit/ 64x16-bit/ 32x32-bit), 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 Common Settings

Figure 2 shows the flow of the general settings of SCB. Specific settings for each protocol are described in later sections.

Figure 2. Flow of General Settings of SCB



Initialize internal clocks related to SCB.
For details, see the Architecture TRM.

Configure the ports for direction (input, output, or High-Z), drive mode, and function.
For details, see the Architecture TRM.

Configure system interrupts for priority level, enable, and interrupt mask.
For details, see the Architecture TRM.

Initialize SCB configurations for the used protocol.

Program the procedure of transmitting and receiving.

The Initialize Clock block sets the peripheral clock (PCLK) input to the SCB. This setting is configured by the clock system in the MCU.

The Configure Port block configures the connection of external pins used for communication with the SCB. The Configure System Interrupt block configures the interrupt settings used for communicating as a transmission interrupt or a receive interrupt.

For details, see the Architecture TRM of each block. The Initialize SCB block initializes the registers of the SCB for the protocol used. The Communication Program block describes the control procedure when the SCB transmits or receives data.

4 SPI Setting Procedure Example

This section shows an example of using SPI. 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.

4.1 Master Mode

This example configures SCB in Motorola SPI Master mode and transmits one word (16-bits) of data and receives one word of data from the SPI Slave.

<Use case>

- SCB Mode = Motorola SPI Master mode
- SCB Channel = 1
- PCLK (Peripheral Clock) = 4 MHz
- Bit rate = 1 Mbps
- Tx/Rx data length = 16 bits
- Tx/Rx FIFO = Yes (16-bit FIFO data elements)
- Rx interrupt = Enable
- Used ports

  SCLK : SCB1_CLK (P18.2)

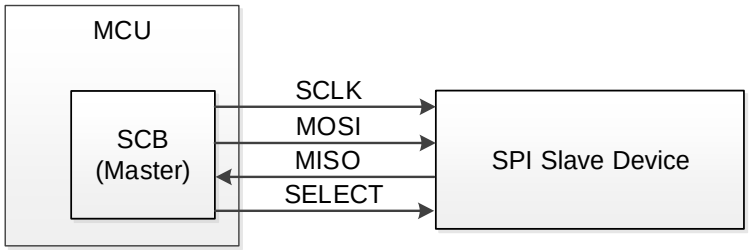
  MOSI : SCB1_MOSI (P18.1)

  MISO : SCB1_MISO (P18.0)

  SELECT : SCB1_SEL0 (P18.3)
- 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 3 shows the example of connection between the SCB and another SPI device.

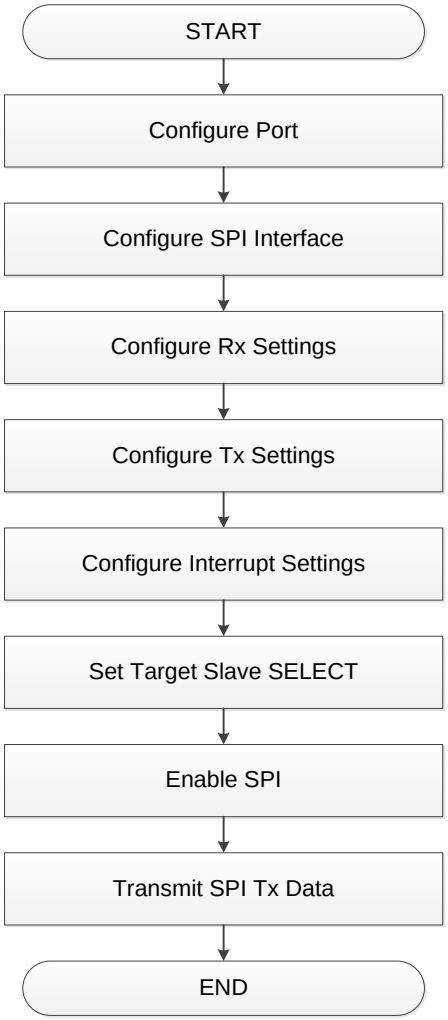
Figure 3. Example of SPI (Master Mode) Communication



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 4 shows the setting procedure and operation example for Master mode.

Figure 4. SPI Master Mode Operation



Configure ports for SPI Master mode. SCLK, MOSI, SELECT are output. MISO is input. See Table 1.

Configure SCB register for SPI Master mode. See Table 2.

Configure Rx settings of SCB register. See Table 3.

Configure Tx settings of SCB register. See Table 4.

Configure interrupt settings of SCB register to allow Rx trigger. See Table 5.

Set target to slave SELECT0 (SSEL[0]).
`SCB_SPI_CTRL.SSEL = “0”`

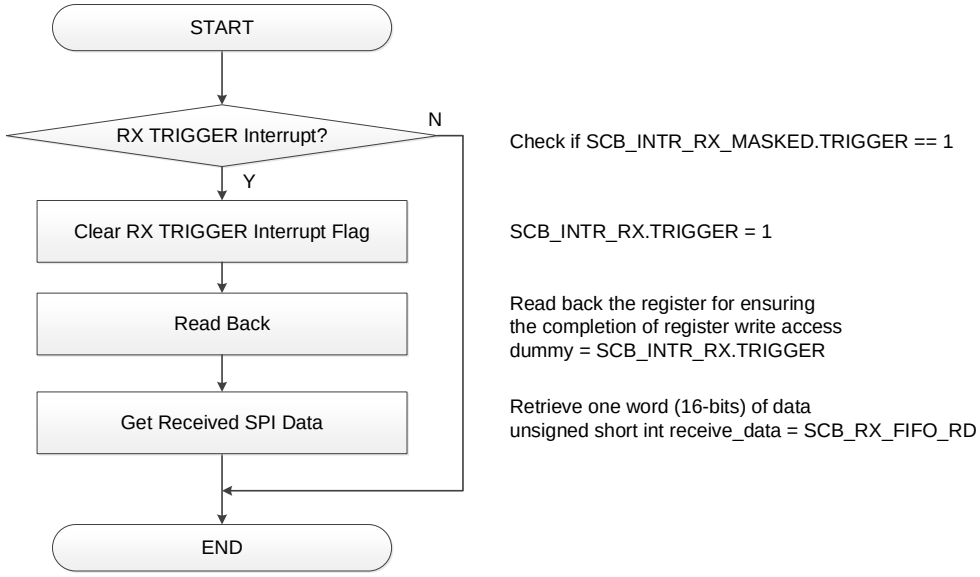
Enable SCB block. When enabled, SCB functions as the configured communication protocol (SPI in this case).
`SCB_CTRL.ENABLED = “1”`

Write the 16-bit data to be transmitted to SCB_TX_FIFO_WR register. Based on Tx setting in Table 4, the SCB will start transmitting as soon two or more bytes are written to the FIFO.
`SCB_TX_FIFO_WR = transmit_data`

After the common settings such as ports and SPI interface configurations, enable the SCB block by setting the enable bit (`SCB_CTRL.ENABLED`). The software sets the transmission data to the Tx FIFO; then the SPI controller transmits the data to the Slave device. If the number of bytes received exceeds the Rx FIFO threshold level (`SCB_RX_FIFO_CTRL.TRIGGER_LEVEL`), the SPI controller notifies the receive interrupt to the CPU. The software can then read the received data from the Rx FIFO.

Figure 5 shows an example of SPI receiver interrupt handling.

Figure 5. SPI Master Mode Interrupt



When the SPI controller stores data in the Rx FIFO, the SPI controller can notify a receive interrupt to the CPU. When the CPU detects a receive interrupt, the software can get the received data from the Rx FIFO. If the cause of an interrupt is not TRIGGER, you should check for other interrupt causes or clear all interrupts.

Note: Procedures for causes other than the TRIGGER interrupt should be handled by the application.

4.1.1 Configure Ports

This section explains an example of the port setting used in SPI Master mode. In this mode, SCLK, MOSI, MISO, and SELECT are used as interface signals. Each signal is assigned to the port number as follows:

SCLK : SCB1_CLK (P18.2)

MOSI : SCB1_MOSI (P18.1)

MISO : SCB1_MISO (P18.0)

SELECT : SCB1_SEL0 (P18.3)

Table 1 shows an example of the port configuration in SPI Master mode.

SCLK, MOSI, and SELECT are configured for the output port, and MISO is configured for the input port with the `GPIO_PRT18_CFG.DRIVE_MODEx` register and `GPIO_PRT18_CFG.IN_ENx` register. The pin functions are determined by the `HSIOM_PRT18.PORT_SEL0.IOx_SEL` register.

Table 1. SPI (Master Mode): Example of Port Configuration

Register	Port Configuration				Remark
	SCLK (*x*=2)	MOSI (*x*=1)	MISO (*x*=0)	SELECT (*x*=3)	
`GPIO_PRT18_CFG.DRIVE_MODEx`	6	6	0	6	0: High-Z. 6: Strong Drive Output.
`GPIO_PRT18_CFG.IN_ENx`	0	0	1	0	0: Input buffer disabled. 1: Input buffer enabled.
`HSIOM_PRT18.PORT_SEL0.IOx_SEL`	19	19	19	19	19: ACT#7. For the SCB function of the pin, see the datasheet.

Note: Bits that are not listed in Table 1 have default values. For default values, see the respective registers in the Registers TRM.

4.1.2 Configure SPI Interface Registers

This section explains an example of setting SPI registers used in SPI Master mode. The following SPI registers are used: interface configuration register, Rx and Tx control register, and interrupt register.

Table 2 shows an example of the SPI interface configuration in SPI Master mode. These registers configure the SPI interface in communication mode or clock and data. In this case, this SPI interface behaves in Motorola mode, SPI Master mode, and 16-bit FIFO data elements.

Table 2. SPI (Master Mode): Example of SPI Interface Configuration

Register	Bit	Value	Remark
SCB_CTRL	MODE	1	SPI mode
	OVS	3	Four oversampling (OVS + 1). This OVS bit is used for determining the bit rate as shown in section 4.1.3. Bit Rate Setting.
	MEM_WIDTH	1	16-bit FIFO data elements
SCB_SPI_CTRL	MODE	0	SPI_MOTOROLA mode
	MASTER_MODE	1	Master mode
	SSEL_POLARITY0	0	SELECT pin to the Slave is LOW/'0' active.
	CPHA	0	When the clock is inactive, the level is LOW. MOSI is driven on the falling edge of SCLK. MISO is captured on the rising edge of SCLK.
	CPOL	0	
	LATE_MISO_SAMPLE	1	The alternate clock edge is used (which comes half a SPI SCLK period later).
	SSEL_CONTINUOUS	0	SCLK is generated when the SPI Master is enabled and data is transmitted.
	SELECT_PRECEDE	0	Used only in SPI Texas Instruments mode.
	SCLK_CONTINUOUS	0	Disables the Master to generate a continuous SCLK regardless of whether there is data to send.

Note: Bits that are not listed in Table 2 have default values. For default values, see the respective registers in the Registers TRM.

Table 3 shows an example of the Rx configuration in SPI Master mode. These registers configure the receiver control settings. In this case, the received data format is Most Significant Bit (MSb) first, and the data width is 16 bits.

Table 3. SPI (Master Mode): Example of Rx Configurations

Register	Bit	Value	Remark
SCB_RX_CTRL	MSB_FIRST	1	MSb first.
	MEDIAN	0	Disables the digital 3-tap median filter to be applied to the input of Rx FIFO to filter glitches on the line.
	DATA_WIDTH	15	Width of Rx data =16.
SCB_RX_FIFO_CTRL	TRIGGER_LEVEL	1	Interrupt occurs when there are more than two entries in the Rx FIFO.

Note: Bits that are not listed in Table 3 have default values. For default values, see the respective registers in the Registers TRM.

Table 4 shows an example of the Tx configuration in SPI Master mode. These registers configure the transmitter control settings. In this case, the transmitter data format is MSb first, and the data width is 16 bits.

Table 4. SPI (Master Mode): Example of Tx Configurations

Register	Bit	Value	Remark
SCB_TX_CTRL	MSB_FIRST	1	MSb first.
	DATA_WIDTH	15	Width of Tx data =16.

Note: Bits that are not listed in Table 4 have default values. For default values, see the respective registers in the Registers TRM.

Table 5 shows an example of the interrupt configuration in SPI Master mode. These registers configure the interrupt mask settings. In this case, when the Rx trigger occurs, the receiver interrupt occurs and the SCB notifies the CPU with the interrupt controller.

Table 5. SPI (Master Mode): Example of Interrupt Configurations

Register	Bit	Value	Remark
SCB_INTR_TX_MASK	-	0	Tx interrupt is masked.
SCB_INTR_RX_MASK	TRIGGER	1	Allow Rx trigger for interrupt.
SCB_INTR_M_MASK	-	0	Master interrupt is masked.
SCB_INTR_S_MASK	-	0	Slave interrupt is masked.
SCB_INTR_SPI_EC_MASK	-	0	External Clock interrupt is masked.

Note: Bits that are not listed in Table 5 have default values. For default values, see the respective registers in the Registers TRM.

4.1.3 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 TRM.

4.2 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.

<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 = Yes (16-bit FIFO data elements)
- Tx/Rx interrupts = Enable
- Used ports

  SCLK : SCB1_CLK (P18.2)

  MOSI : SCB1_MOSI (P18.1)

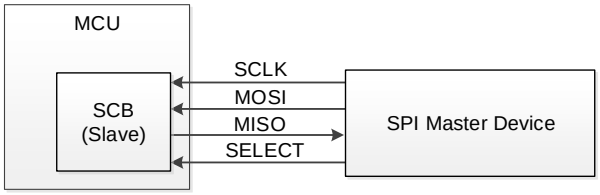
  MISO : SCB1_MISO (P18.0)

  SELECT : SCB1_SEL0 (P18.3)

- MISO data is driven on a falling edge of SCLK.
- MOSI data is captured on a rising edge of SCLK

Figure 6 shows the example of a connection between the SCB and another SPI device.

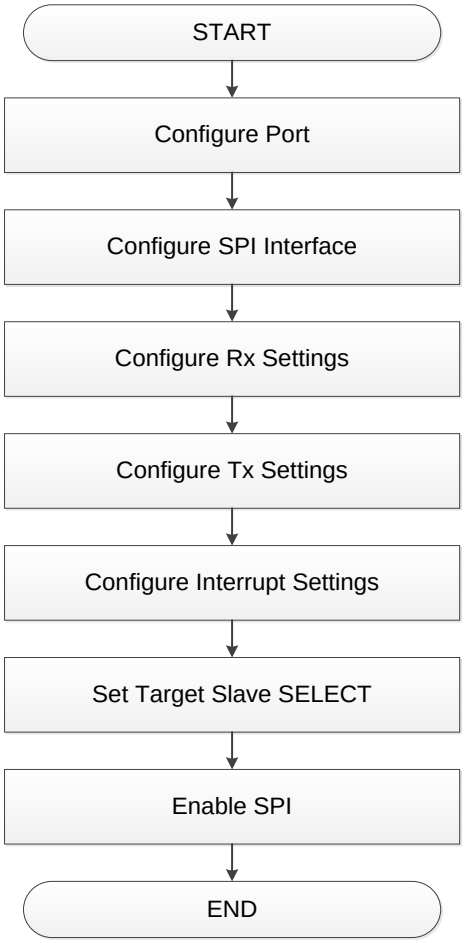
Figure 6. Example of SPI (Slave Mode) Communication Connection



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 7 shows the setting procedure and operation example for Slave mode.

Figure 7. SPI Slave Mode Operation



Configure ports for SPI Slave mode. SCLK, MOSI, SELECT are inputs. MISO is output. See Table 6.

Configure SCB register for SPI Slave mode. See Table 7.

Configure Rx settings of SCB register. See Table 8.

Configure Tx settings of SCB register. See Table 9.

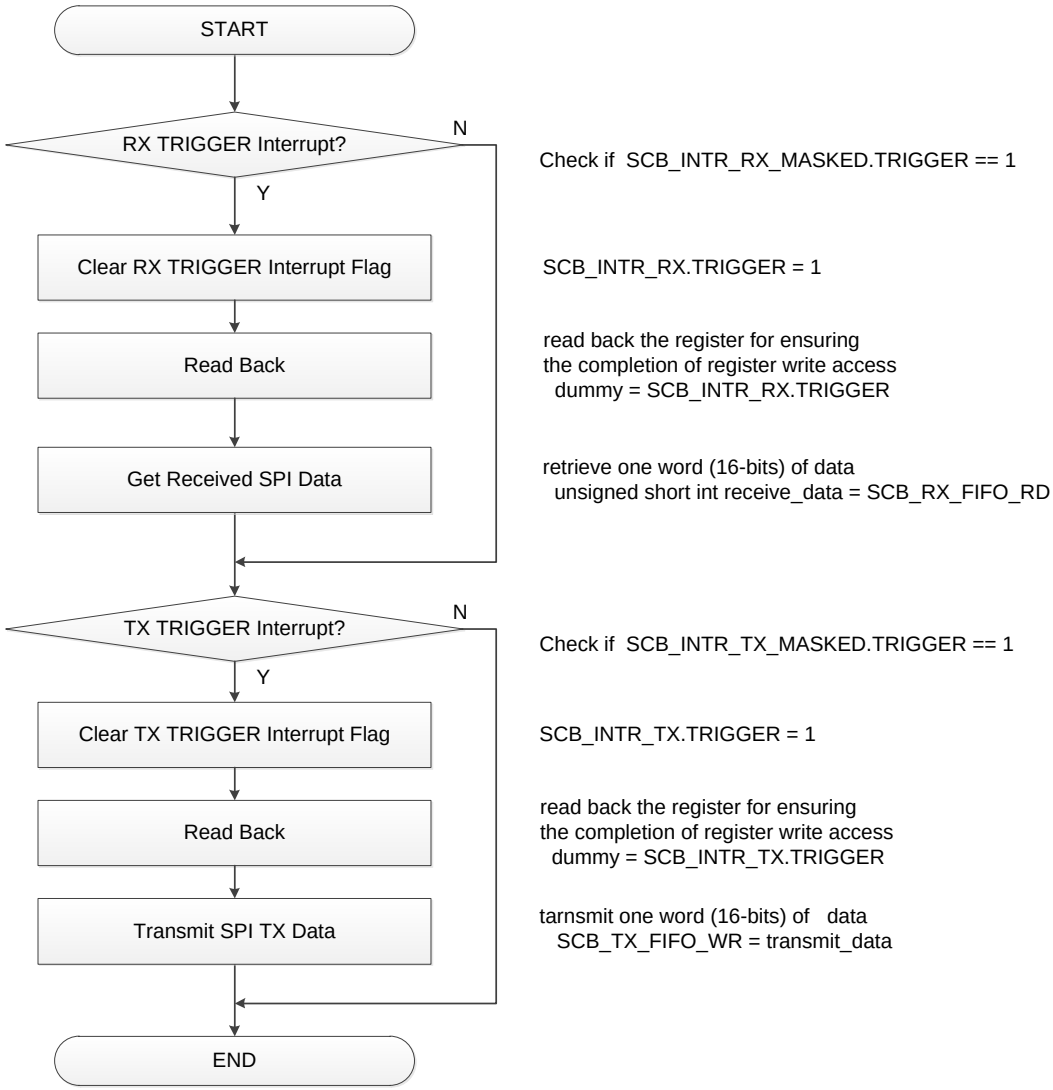
Configure interrupt settings of SCB register to allow Tx and Rx triggers. See Table 10.

Set target to slave SELECT0 (SSEL[0]).
`SCB_SPI_CTRL.SSEL = “0”`

Enable SCB block. When enabled, SCB functions as the configured communication protocol (SPI in this case).
`SCB_CTRL.ENABLED = “1”`

The SCB is configured in SPI Slave mode by these procedures. Received and transmitted interrupts are configured with each threshold level of FIFO. After the SPI controller is enabled (`SCB_CTRL.ENABLE = “1”`), the SPI controller waits for the interrupt trigger. If the SPI controller receives an interrupt, the interrupt handler is called as shown in Figure 8, which shows an example of the SPI controller receiver interrupt handling.

Figure 8. SPI Slave Mode Interrupt



In this case, when more than two bytes of data is stored in the Rx FIFO, the SPI controller can notify a receive interrupt to the CPU. After the CPU detects a receive interrupt, if `SCB_INTR_RX.TRIGGER` is “1”, the software can get two bytes of the received data from the Rx FIFO. On the other hand, when there is no entry data in Tx FIFO, the SPI controller can also notify a transmission interrupt to the CPU. If `SCB_INTR_TX.TRIGGER` is “1”, the software can transmit the data via the Tx FIFO.

Note: Procedures for causes other than the TRIGGER interrupt should be handled by the application.

4.2.1 Configure Ports

This section explains an example of the port setting used in SPI Slave mode. In this mode, SCLK, MOSI, MISO, and SELECT are used as interface signals. Each signal is assigned to the port number as follows:

SCLK : SCB1_CLK (P18.2)

MOSI : SCB1_MOSI (P18.1)

MISO : SCB1_MISO (P18.0)

SELECT : SCB1_SEL0 (P18.3)

Table 6 shows an example of the port configuration in SPI Slave mode.

SCLK, MOSI, and SELECT are configured for input ports, and MISO is configured for the output port with the GPIO_PRT18_CFG.DRIVE_MODE*x* register and the GPIO_PRT18_CFG.IN_EN*x* register. The pin functions are determined by the HSIOM_PRT18.PORT_SEL0.IO*x*_SEL register.

Table 6.SPI (Slave Mode): Example of PORT Configurations

Register	Port Configuration				Remark
	SCLK (*x*=2)	MOSI (*x*=1)	MISO (*x*=0)	SELECT (*x*=3)	
GPIO_PRT18_CFG.DRIVE_MODEx	0	0	6	0	0: High-Z. 6: Strong Drive Output.
GPIO_PRT18_CFG.IN_ENx	1	1	0	1	0: Input buffer disabled. 1: Input buffer enabled.
HSIOM_PRT18.PORT_SEL0.IOx_SEL	19	19	19	19	19: ACT#7. For the SCB function of the pin, see the datasheet.

Note: Bits that are not listed in the table above have their default values. For default values, refer the respective registers in the Registers TRM.

4.2.2 Configure SPI Interface Registers

This section explains an example of the SPI registers setting in SPI Slave mode. The following SPI registers are used: Interface configuration register, Rx and Tx control register, and Interrupt register.

Table 7 shows an example of the SPI interface configuration in SPI Slave mode. These registers configure the SPI interface with a communication mode or a format of clock and data. In this case, the SPI interface behaves in Motorola mode, SPI Slave mode, and 16-bit FIFO data elements.

Table 7. SPI (Slave Mode): Example of SPI Interface Configurations

Register	Bit	Value	Remark
SCB_CTRL	MODE	1	SPI mode.
	OVS	15	No effect in Slave mode.
	MEM_WIDTH	1	16-bit FIFO data elements.
SCB_SPI_CTRL	MODE	0	SPI_MOTOROLA mode.
	MASTER_MODE	0	Slave mode.
	SSEL_POLARITY0	0	SELECT pin to the Slave is LOW/'0' active.
	CPHA	0	When the clock is inactive, the level is LOW. MOSI is driven on the falling edge of SCLK. MISO is captured on the rising edge of SCLK.
	CPOL	0	

Note: Bits that are not listed in Table 7 have default values. For default values, see the respective registers in the Registers TRM.

Table 8 shows an example of the Rx configuration in SPI Slave mode. These registers configure the receiver control settings. In this case, the received data format is MSb first, and the data width is 16 bits.

Table 8. SPI (Slave Mode): Example of Rx Configurations

Register	Bit	Value	Remark
SCB_RX_CTRL	MSB_FIRST	1	MSb first.
	MEDIAN	0	Disables the digital 3-tap median filter to be applied to the input of the Rx FIFO to filter glitches on the line.
	DATA_WIDTH	15	Width of Rx data =16.
SCB_RX_FIFO_CTRL	TRIGGER_LEVEL	1	Interrupt occurs when there are more than two entries in the Rx FIFO.

Note: Bits that are not listed in Table 8 their default values. For default values, see the respective registers in the Registers TRM.

Table 9 shows an example of the Tx configuration in SPI Slave mode. These registers configure the transmitter control settings. In this case, the transmitter data format is MSb first, and the data width is 16 bits.

Table 9. SPI (Slave Mode): Example of Tx Configurations

Register	Bit	Value	Remark
SCB_TX_CTRL	MSB_FIRST	1	MSb first.
	DATA_WIDTH	15	The width of Tx data =16.
SCB_TX_FIFO_CTRL	TRIGGER_LEVEL	1	Interrupt occurs when Tx FIFO is empty.

Note: Bits that are not listed in Table 9 have their default values. For default values, refer the respective registers in the Registers TRM.

Table 10 shows an example of the interrupt configuration in SPI Slave mode. These registers configure the interrupt mask settings. In this case, when an Rx trigger occurs, the interrupt occurs and the SCB notifies the CPU with the interrupt controller.

Table 10. SPI (Slave Mode): Example of Interrupt Configurations

Register	Bit	Value	Remark
SCB_INTR_TX_MASK	TRIGGER	1	Allow interrupt to be triggered based on the TRIGGER_LEVEL setting of SCB_TX_FIFO_CTRL.
SCB_INTR_RX_MASK	TRIGGER	1	Allow interrupt to be triggered based on the TRIGGER_LEVEL setting of SCB_RX_FIFO_CTRL.
SCB_INTR_M_MASK	-	0	Master interrupt mask is masked.
SCB_INTR_S_MASK	-	0	Slave interrupt mask is masked.
SCB_INTR_SPI_EC_MASK	-	0	Externally clocked SPI interrupt is masked.

Note: Bits that are not listed in Table 10 have default values. For default values, see the respective registers in the Registers TRM.

5 UART Setting Procedure Example

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

5.1 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.

<Use case>

- SCB Mode = Standard UART
- SCB Channel = 3
- PCLK = 40MHz
- Baud Rate = 115,200 bps
- Data width = 8 bits
- Parity = None
- Stop Bits = 1

- Flow Control = None
- TX/RX FIFO = Yes
- RX interrupt = Enable
- Used ports

  Tx : SCB3_TX (P13.1)

  Rx : SCB3_RX (P13.0)

An example TX-RX connection between the SCB and the external UART device is shown in Figure 9. In this example, flow control signals RTS and CTS are not used.

Figure 9. Example of UART Communication Connection

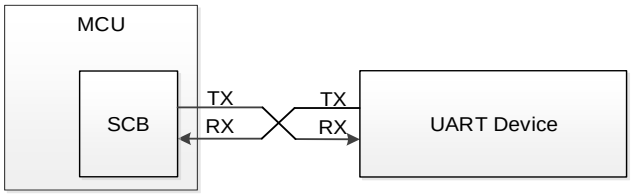
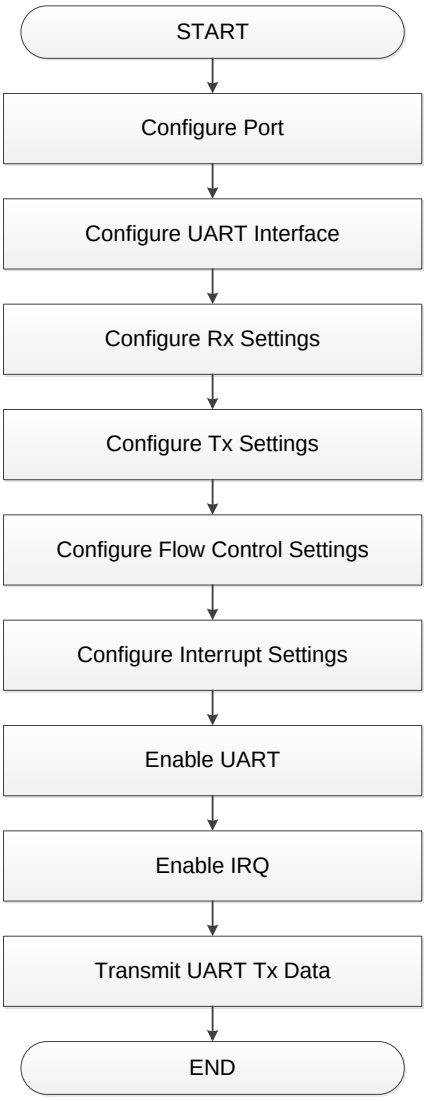


Figure 10 shows the setting procedure and operation example for UART.

Figure 10. UART Operation



Configure ports for UART mode. Tx is output. Rx is input. See Table 11.

Configure SCB register for the UART mode. See Table 12.

Configure Rx settings of the SCB register. See Table 13.

Configure Tx settings of the SCB register. See Table 14.

Configure flow control of the SCB register. See Table 15.

Configure interrupt settings of the SCB register to enable interrupts of data reception. See Table 16.

Enable SCB block. When enabled, SCB functions as the configured communication protocol (UART in this case). SCB_CTRL.ENABLED = “1”

Enable IRQ in NVIC
For details, see the Architecture TRM.

Write the data to be transmitted over UART into SCB TX FIFO. SCB_TX_FIFO_WR = transmit_data

After the general configuration (clock, port, interrupt, and so on), in this case, SCB registers are configured for UART. The software then transmits the data over UART by writing to the TX FIFO. If the number of received data exceeds the threshold of Rx FIFO (`SCB_RX_FIFO_CTRL.TRIGGER_LEVEL`), the receive interrupt occurs and the interrupt handler would be called.

Figure 11 shows an example of UART interrupt handling.

Figure 11. UART: Example of Interrupt Flowchart

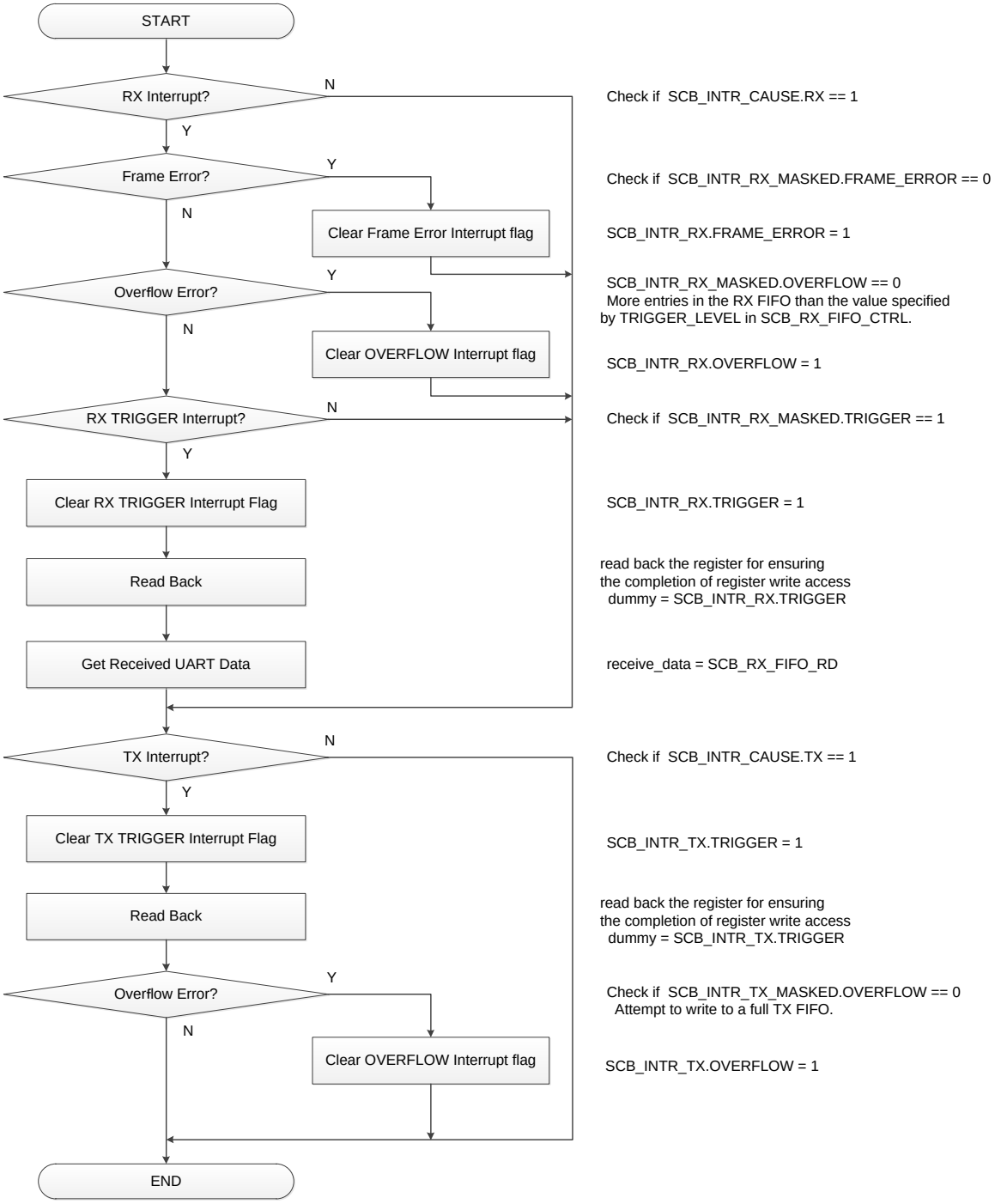


Figure 11 shows the interrupt procedure that will be executed when a TX/RX interrupt occurs. If an Rx interrupt occurs, received errors are checked, the interrupt flag is cleared, and the received data is read. If a Tx interrupt occurs, the interrupt flag is cleared, and the overflow error is checked.

Note: The error procedures should be handled by the application.

5.1.1 Configure Ports

This section provides an example of the port setting used in UART mode. In this mode, Tx and Rx are used as interface signals. Each signal is assigned to the port number as follows:

Tx : SCB3_TX (P13.1)

Rx : SCB3_RX (P13.0)

Table 11 shows an example of port configuration in UART mode.

Tx is configured for the output port, and Rx is configured for the input port with the `GPIO_PRT13_CFG.DRIVE_MODEx` register and `GPIO_PRT13_CFG.IN_ENx` register. Pin functions are determined by the `HSIOM_PRT13.PORT_SEL0.IOx_SEL` register.

Table 11. UART: Example of PORT Configurations

Register	Port Configuration		Remark
	Tx (*x*=1)	Rx (*x*=0)	
`GPIO_PRT13_CFG.DRIVE_MODEx`	6	0	0: High-Z 6: Strong Drive Output
`GPIO_PRT13_CFG.IN_ENx`	0	1	0: Input buffer disabled 1: Input buffer enabled
`HSIOM_PRT13.PORT_SEL0.IOx_SEL`	17	17	17: ACT#5, for SCB function of the pin, Refer datasheet.

Note: Bits that are not listed in Table 11 have default values. For default values, see the respective registers in the Registers TRM.

5.1.2 Configure UART Interface Registers

This section explains an example of the UART registers setting in standard UART mode. The following UART registers are used: interface configuration register, Rx and Tx control register, and interrupt register.

Table 12 shows an example of the UART interface configuration. These registers configure the UART control settings in communication mode or as an oversampling value. In this case, the UART interface behaves in standard UART mode, 16 oversampling, and 8-bit FIFO data elements.

Table 12. UART: Example of UART Interface Configurations

Register	Bit	Value	Remark
`SCB_CTRL`	`MODE`	2	UART mode
	`OVS`	15	16 oversampling (OVS + 1)
	`MEM_WIDTH`	0	8-bit FIFO data elements
`SCB_UART_CTRL`	`MODE`	0	Standard UART mode

Note: Bits that are not listed in Table 12 have default values. For default values, see the respective registers in the Registers TRM.

Table 13 shows an example of the Rx configuration in UART mode. These registers configure the receiver control settings. In this case, the received data format is LSb first, the data width is 8 bits, stop bit is 1, and data parity is none.

The received interrupt threshold (`SCB_RX_FIFO_CTRL.TRIGGER_LEVEL`) is set to “0”, which means that when one byte is received, an Rx interrupt should occur.

Table 13 UART: Example of Rx Configurations

Register	Bit	Value	Remark
SCB_UART_RX_CTRL	POLARITY	0	Non-inversion.
	MP_MODE	0	Multi-processor mode disabled.
	DROP_ON_PARITY_ERROR	0	Even if parity check fails, the received data is sent to the Rx FIFO.
	DROP_ON_FRAME_ERROR	0	Even if an error is detected in a start or stop period, the received data is sent to the Rx FIFO.
	STOP_BITS	1	Stop bits is STOP_BITS + 1 (= 2 of half bits).
	PARITY	0	Parity not used.
	PARITY_ENABLED	0	Parity disabled.
SCB_RX_CTRL	MSB_FIRST	0	Least significant bit (LSb) first.
	MEDIAN	0	Median filter disabled.
	DATA_WIDTH	7	DATA_WIDTH + 1 (=8) is the expected number of bits in the received data frame.
SCB_RX_MATCH	ADDR	0	Slave device address =0.
	MASK	0	Slave device address mask =0.
SCB_RX_FIFO_CTRL	TRIGGER_LEVEL	0	When one byte received, the Rx interrupt should occur.

Note: Bits that are not listed in Table 13 have default values. For default values, see the respective registers in the Registers TRM.

Table 14 shows an example of the Tx configuration in UART mode. These registers configure the transmitter control settings. In this case, the transmitter data format is LSb first, the data width is 8 bits, stop bit is 1, and data parity is none. The transmitted interrupt threshold (SCB_RTX_FIFO_CTRL.TRIGGER_LEVEL) is set to “0”, which means that when one byte transmitted, a Tx interrupt should occur.

Table 14. UART: Example of Tx Configurations

Register	Bit	Value	Remark
SCB_UART_TX_CTRL	RETRY_ON_NACK	0	Not used.
	STOP_BITS	1	Stop bits is STOP_BITS + 1 (= 2 of halve bit).
	PARITY	0	Parity not used.
	PARITY_ENABLED	0	Parity disabled.
SCB_TX_CTRL	MSB_FIRST	0	Least significant bit (LSb) first.
	DATA_WIDTH	7	DATA_WIDTH + 1 (=8) is the number of bits in a transmitted data frame.
	OPEN_DRAIN	0	Normal operation mode.
SCB_TX_FIFO_CTRL	TRIGGER_LEVEL	0	When one byte transmitted, Tx interrupt should occur.

Note: Bits that are not listed in Table 14 have default values. For default values, see the respective registers in the Registers TRM.

Table 15 shows an example of the flow control configuration in UART mode. These registers configure the data flow control settings. In this use case, CTS and RTS are not used.

Table 15. UART: Example of Flow Control Configurations

Register	Bit	Value	Remark
SCB_UART_FLOW_CTRL	CTS_ENABLED	0	CTS disabled.
	CTS_POLARITY	0	Not used.
	RTS_POLARITY	0	Not used.
	TRIGGER_LEVEL	0	Not used.

Note: Bits that are not listed in Table 15 have default values. For default values, see the respective registers in the Registers TRM.

Table 16 shows an example of the interrupt configuration in UART mode. These registers configure the interrupt mask settings. Rx interrupt trigger, Tx interrupt trigger, Tx transmission done, and various alarms can be configured to be enabled. In this case, when the Tx TRIGGER and UART_DONE interrupts occur, the SCB notifies to the CPU with the interrupt controller.

Table 16. UART: Example of Interrupt Configurations

Register	Bit	Value	Remark
SCB_INTR_RX_MASK	-	0xB61	The following interrupt factor is enabled: TRIGGER, OVERFLOW, UNDERFLOW, FRAME_ERROR, PARITY_ERROR, BREAK_DETECT.
SCB_INTR_TX_MASK	-	0x221	The following interrupt factor is enabled: TRIGGER, OVERFLOW, UART_DONE.

Note: Bits that are not listed in Table 16 have their default values. For default values, refer the respective registers in the Registers TRM.

5.1.3 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:

Calculation Example

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

You can use the specified for 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]
```

6 I²C Setting Procedure Example

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

6.1 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.

<Use case>

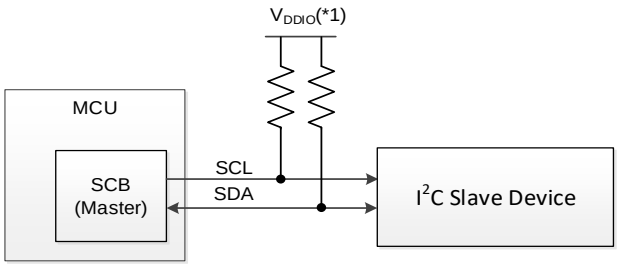
- SCB Mode = I²C Master mode
- SCB Channel = 0
- PCLK = 2 MHz
- Bit rate = 100 kbps
- 7-bit Slave address = 0x8 (for another I²C device)
- MSb first
- Tx/Rx FIFO = Yes
- Tx/Rx interrupt = Disabled
- Analog filter is enabled and digital filter = Disabled
- Used ports

  SCL : SCB0_SCL (P1.0)

  SDA : SCB0_SDA (P1.1)

Figure 12 shows the example of connection between the SCB and another I²C Slave device.

Figure 12. Example of I²C (Master Mode) Communication

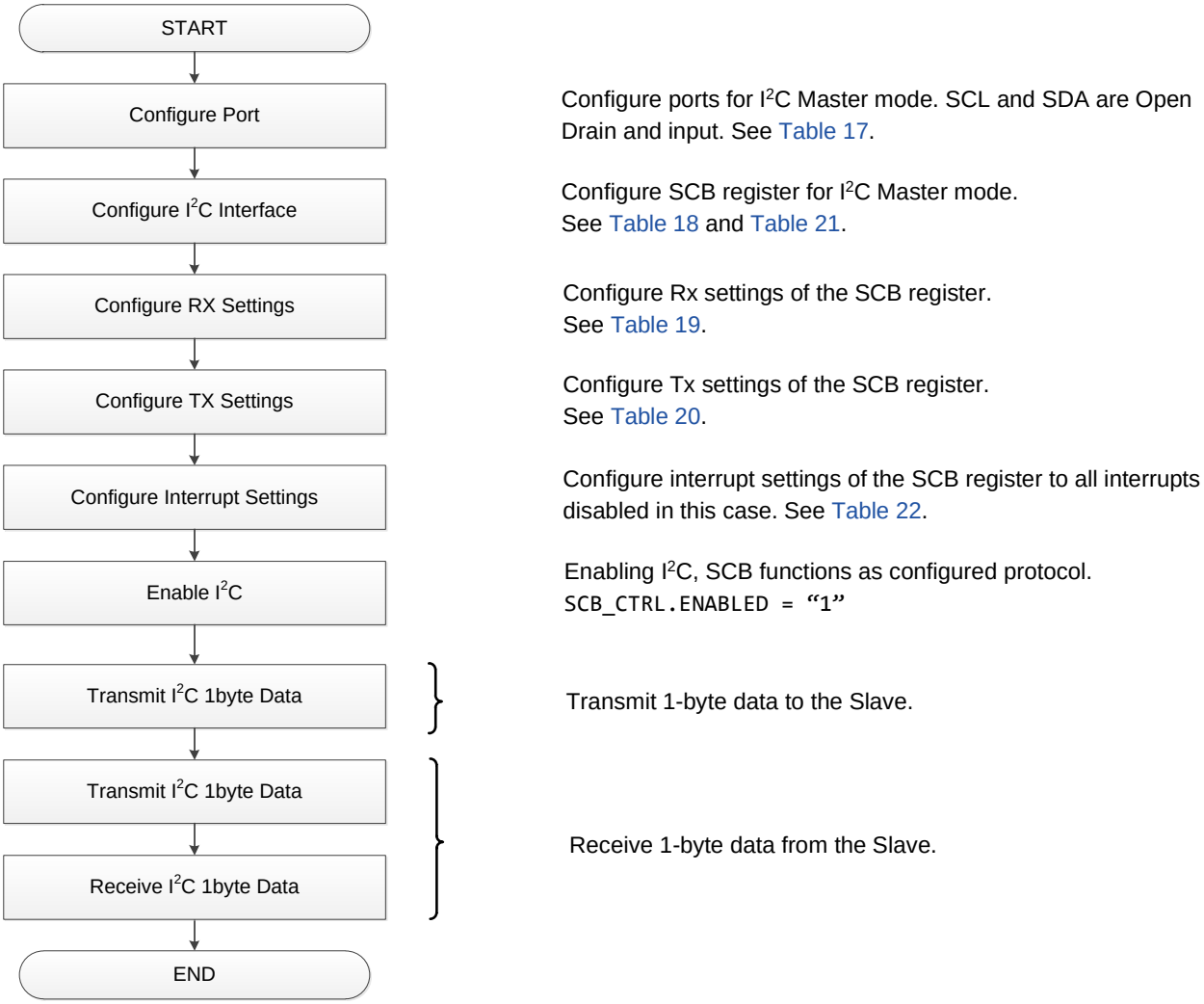


Note (*1) For V_{DDIO} value, see the datasheet (see Related Documents).

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 V_{DDIO} via external pull up register.

Figure 13 shows setting procedure and operation example for I²C Master mode.

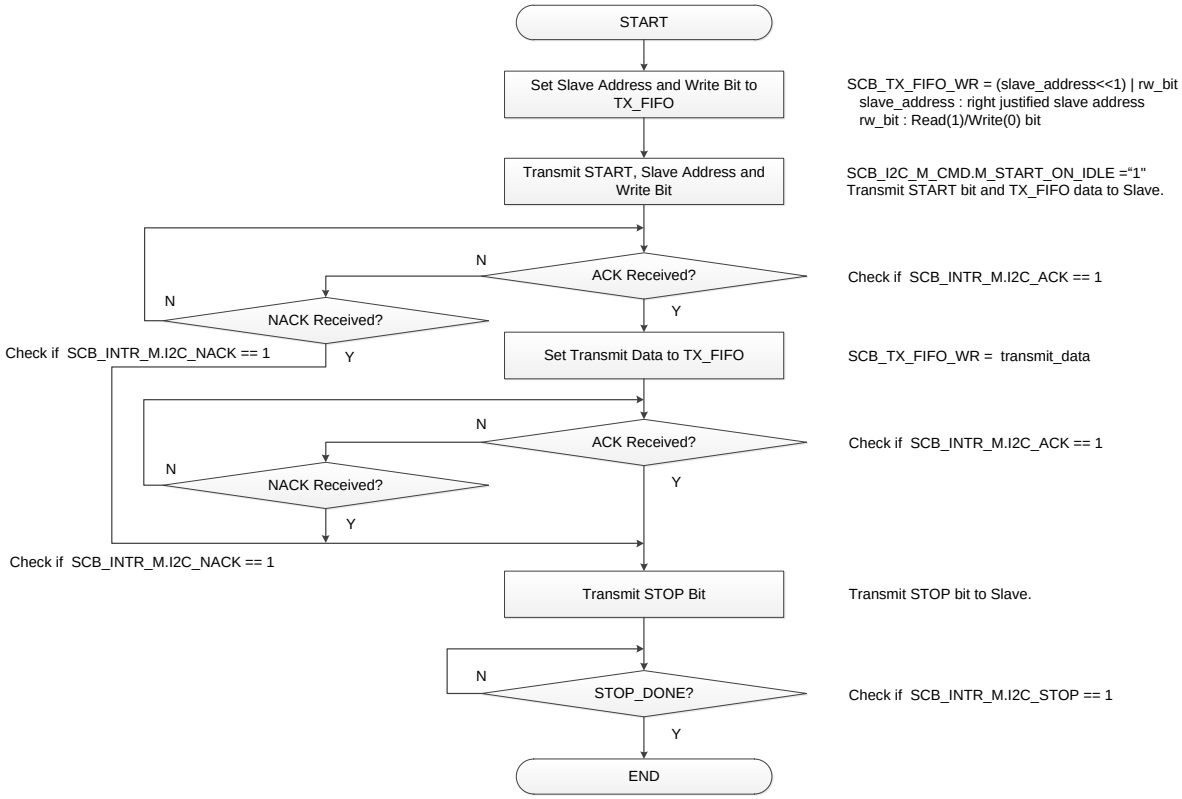
Figure 13. I²C Master Mode Operation



After the common settings (clock, port, interrupt, and so on), the interface register of the SCB is configured for I²C. If the `SCB_CTRL.ENABLE` register is set "1", the SCB is ready to transmit and receive the data. Then, the I²C Master writes one byte to the slave and reads one byte from the slave.

Figure 14 shows a procedure example of I²C Master transmission.

Figure 14. Example for I²C Master Data Transmission

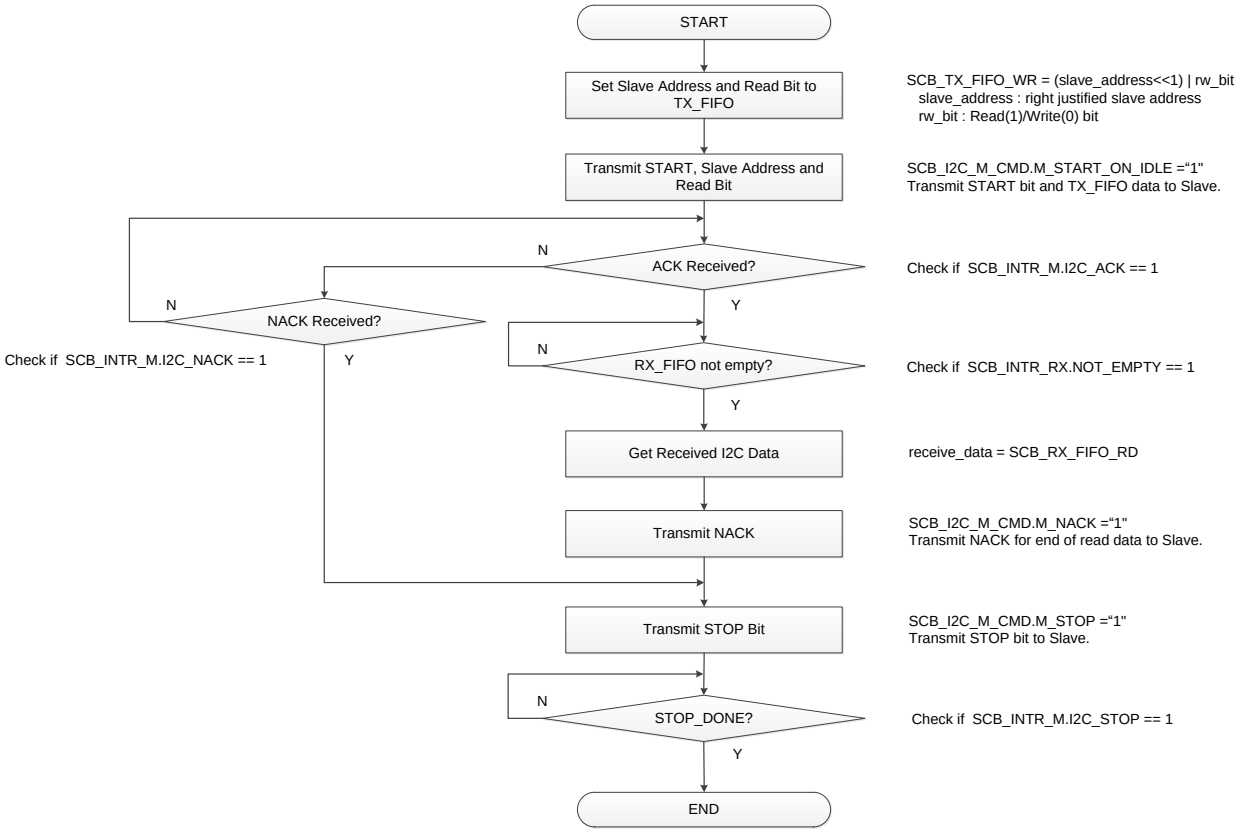


After transmitting the first byte consisting of the slave address and the write bit, the firmware polls the `SCB_INTR_M.I2C_ACK/NACK` bit until an ACK/NACK is received from the slave. If ACK is received, the data byte is transmitted before sending the stop bit. If the NACK is received , a STOP is issued on the I2C bus to terminate the bus transfer.

Note: Handling of I2C bus errors/arbitration loss is not illustrated in the above example.

Figure 15 shows a procedure example of I²C Master reception.

Figure 15. Example for I²C Master Mode Reception



After transmitting the first byte consisting of the slave address and the read bit, the firmware polls the SCB_INTR_M.I2C_ACK/NACK bit until an ACK/NACK is received from the Slave. Firmware then waits for the data to be available in the RX FIFO (RX_FIFO not empty), and then transmits NACK and STOP to the slave to finish the read operation.

Note: Handling of I2C bus errors/arbitration loss is not illustrated in the above example.

6.1.1 Configure Ports

This section explains an example of the port setting used in I²C Master mode. In this mode, SCL and SDA are used as interface signals. Each signal is assigned to the port number as shown follows:

SCL : SCB0_SCL (P1.0)
SDA : SCB0_SDA (P1.1)

Table 17 shows an example of the port configuration in I²C Master mode.

SCL and SDA are configured for open-drain and input port with the `GPIO_PRT1_CFG.DRIVE_MODEx` register and the `GPIO_PRT1_CFG.IN_ENx` register. The pin functions are determined by the `HSIOM_PRT1.PORT_SEL0.IOx_SEL` register.

Table 17. I²C (Master Mode): Example of Port Configurations

Register	Port Configuration		Remark
	SCL (*x*=0)	SDA (*x*=1)	
`GPIO_PRT1_CFG.DRIVE_MODEx`	4	4	4: Open Drain, Drives Low
`GPIO_PRT1_CFG.IN_ENx`	1	1	1: Input buffer enabled
`HSIOM_PRT1.PORT_SEL0.IOx_SEL`	14	14	14: DS#2

Note: Bits that are not listed Table 17 have default values. For default values, refer the respective registers in the Registers TRM.

6.1.2 Configure I²C Interface Registers

This section explains an example of the I²C register setting used in I²C Master mode. The following I²C registers are used: interface configuration register, Rx and Tx control register, and interrupt register.

Table 18 shows an example of the I²C interface configuration in I²C Master mode. These registers determine the behavior of the SCB. In this case, the SCB is configured in I²C Master mode. This is not a multi-Master mode, so the `SCB_I2C_CTRL.SLAVE_MODE` bit set to disable.

`SCB_I2C_CTRL.HIGH_PHASE_OVS` and `SCB_I2C_CTRL.LOW_PHASE_OVS` determine the I²C bit rate. For more details, see Bit Rate Setting.

Table 18. I²C (Master Mode): Example of I²C Interface Configurations

Register	Bit	Value	Remark
SCB_CTRL	MODE	0	I²C mode
	ADDR_ACCEPT	0	Slave address not accepted
	MEM_WIDTH	0	8-bit FIFO data elements
	EC_AM_MODE	0	Internally clocked mode
SCB_I2C_CTRL	MASTER_MODE	1	Master mode
	SLAVE_MODE	0	Disabled
	HIGH_PHASE_OVS	9	HIGH phase oversampling factor is 10
	LOW_PHASE_OVS	9	LOW phase oversampling factor is 10

Note: Bits that are not listed in Table 18 have default values. For default values, see the respective registers in the Registers TRM.

Table 19 shows an example of the Rx Configuration in I²C Master mode. These registers configure the receiver control settings. In this case, the received data format is MSb first, the data width is 8 bits, and digital median filter is disabled.

Table 19. I²C (Master Mode): Example of Rx Configurations

Register	Bit	Value	Remark
SCB_RX_CTRL	MSB_FIRST	1	MSb first
	DATA_WIDTH	7	DATA_WIDTH + 1 (=8) is the expected number of bits in received data frame.
	MEDIAN	0	Digital median filter is disabled.

Note: Bits that are not listed in Table 19 have default values. For default values, see the respective registers in the Registers TRM.

Table 20 shows an example of the Tx configuration in I²C Master mode. These registers configure the transmitter control settings. In this case, the transmitter data format is MSb first, the data width is 8 bits, and OPEN_DRAIN mode is set.

Table 20. I²C (Master Mode): Example of Tx Configurations

Register	Bit	Value	Remark
SCB_TX_CTRL	MSB_FIRST	1	MSb first.
	DATA_WIDTH	7	DATA_WIDTH + 1 (=8) is the number of bits in a transmitted data frame.
	OPEN_DRAIN	1	Open drain operation mode.

Note: Bits that are not listed in Table 20 have default values. For default values, see the respective registers in the Registers TRM.

Table 21 shows an example of the analog filter configuration in I²C Master mode. These registers configure the analog filter settings to remove glitches. In this case, the filters of SDA_IN and SCLK_IN are enabled; SDA_OUT is disabled.

Table 21. I²C (Master Mode): Example of Analog Filter Configurations

Register	Bit	Value	Remark
SCB_I2C_CFG	SDA_IN_FILT_SEL	1	SDA input filter is enabled.
	SCL_IN_FILT_SEL	1	SCL input filter is enabled.
	SDA_OUT_FILT_SEL	0	SDA output filter is disabled.

Note: Bits that are not listed in Table 21 have default values. For default values, see the respective registers in the Registers TRM.

Table 22 shows an example of the interrupt configuration in I²C Master mode. These registers configure the interrupt mask settings. In this case, Tx and Rx interrupt is not used, because the software polls the status bits of the SCB_INTR_M register.

Table 22. I²C (Master Mode): Example of Interrupt Configurations

Register	Bit	Value	Remark
SCB_INTR_I2C_EC_MASK	-	0	Externally clocked I²C interrupt request is masked.
SCB_INTR_M_MASK	-	0	Master interrupt is masked.
SCB_INTR_S_MASK	-	0	Slave interrupt is masked.
SCB_INTR_TX_MASK	-	0	Transmitter interrupt is masked.
SCB_INTR_RX_MASK	-	0	Receiver interrupt is masked.

Note: Bits that are not listed in Table 22 have default values. For default values, see the respective registers in the Registers TRM.

6.1.3 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.

6.2 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.

<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 = Yes

- MSb first
- Data width = 8 bits
- Analog filter is enabled and digital filter = Disabled
- Enabled interrupts:

  I2C_ARB_LOST (I^2C Slave arbitration lost)

  I2C_STOP (I^2C STOP event detected)

  I2C_ADDR_MATCH (I^2C Slave address matching)

  I2C_GENERAL (I^2C Slave general call address received)

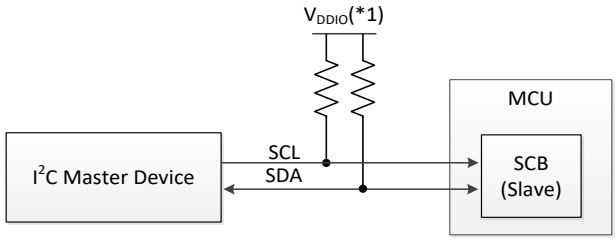
  I2C_BUS_ERROR ((I^2C Slave bus error detected)
- Used ports

  SCL : SCB0_SCL (P1.0)

  SDA : SCB0_SDA (P1.1)

Figure 16 shows the example of the connection between the Slave SCB and another I^2C Master device.

Figure 16. Example of I^2C (Slave Mode) Communication Connection

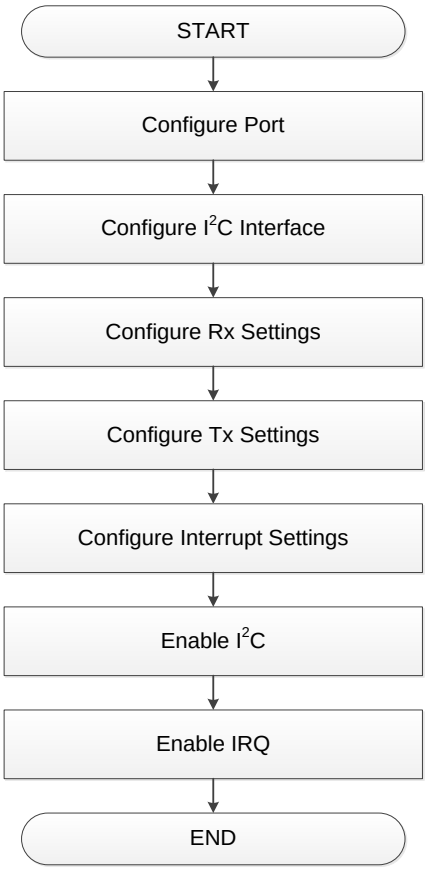


Note (*1) For V_{DDIO} value, see the datasheet (see Related Documents).

In I^2C Slave mode, SCL and SDA signals are connected to another I^2C 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 V_{DDIO} via a resistor.

Figure 17 shows the setting procedure and operation example for I^2C Slave mode.

Figure 17. I²C Slave Mode Operation



Configure ports for I²C Slave mode. SCL is High-Z (input). SDA is Open Drain and input. See Table 23.

Configure SCB register for I²C Slave mode. See Table 24 and Table 27.

Configure Rx settings of SCB register. See Table 25.

Configure Tx settings of SCB register. See Table 26.

Configure interrupt settings of SCB register to allow the Slave to receive the trigger. See Table 28.

Enabling I²C, SCB performs as configured protocol. SCB_CTRL.ENABLED = “1”

Enable IRQ in NVIC
For details, see Architecture TRM.

After the general configuration (clock, port, interrupt, and so on.), the interface register of the SCB is configured for I²C. If the `SCB_CTRL.ENABLE` register is set “1” and interrupt is enabled, the SCB is ready to receive data.

Figure 18 shows an example for I²C Slave reception interrupt.

Figure 18. I²C Slave Mode Reception Interrupt Example

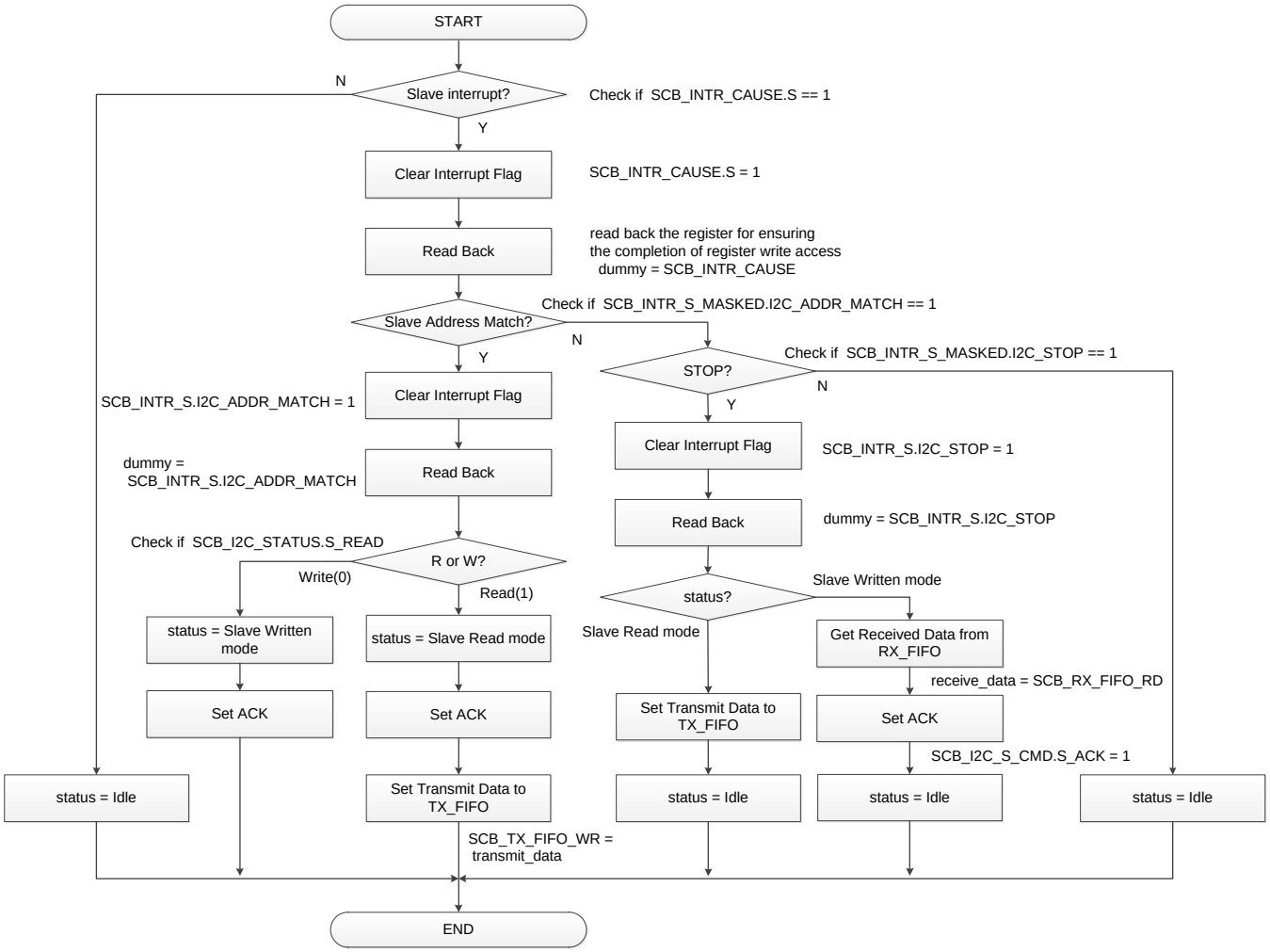


Figure 18 shows the interrupt procedure. After a Slave interrupt (SCB_INTR_CAUSE.S) occurs, the interrupt flag should be cleared, and the software can check whether the received Slave address matches with the configured Slave address. If the received Slave address matches with the configured Slave address, the received data might be the first byte of the I²C format, and then the software can check whether the direction of the data is write (from Master to Slave) or read (from Slave to Master). If the received Slave address does not match and STOP interrupt is TRUE, the flow branches based on the status of the first received byte. If the status is Slave Read mode, set the transmit data to the TX_FIFO register as the response data to the master. If the status is Slave Written mode, the received data may be the written data from the Master; then the software can read out the received data from the `RX_FIFO` register. If the received Slave address does not match and the STOP interrupt is FALSE, the received data may be invalid.

Note: Error handling is not explained in Figure 18 and shall be handled by the application.

6.2.1 Configure Ports

This section explains an example of the port setting used in I²C Slave mode. In this mode, SCL and SDA are used as interface signals. Each signal is assigned to the port number as follows:

SCL : SCB0_SCL (P1.0)

SDA : SCB0_SDA (P1.1)

Table 23 shows an example of the port configuration in I²C Slave mode.

SCL and SDA are configured for open-drain and input port with the GPIO_PRT1_CFG.DRIVE_MODE*x* register and the GPIO_PRT1_CFG.IN_EN*x* register. The port functions are determined by the HSIOM_PRT1.PORT_SEL0.IO*x*_SEL register.

Table 23. I²C (Slave Mode): Example of Port Configurations

Register	Port Configuration		Remark
	SCL (*x*=0)	SDA (*x*=1)	
GPIO_PRT1_CFG.DRIVE_MODEx	0	4	0: High-Z. 4: Open Drain, Drives Low.
GPIO_PRT1_CFG.IN_ENx	1	1	1: Input buffer enabled.
HSIOM_PRT1.PORT_SEL0.IOx_SEL	14	14	14: DS#2.

Note: Bits that are not listed in Table 23 have default values. For default values, see the respective registers in the Registers TRM.

6.2.2 Configure I²C Interface Registers

This section explains an example of the I²C registers setting used in I²C Slave mode. The following registers are used: interface configuration register, Rx and Tx control register, and Interrupt register.

Table 24 shows an example of the I²C interface configuration in I²C Slave mode. These registers determine the behavior of the SCB. In this case, the SCB is configured in the I²C Slave mode by setting SCB_I2C_CTRL.SLAVE_MODE to "1".

Table 24. I²C (Slave Mode): Example of I²C Interface Configurations

Register	Bit	Value	Remark
SCB_CTRL	MODE	0	I²C mode.
	ADDR_ACCEPT	0	Slave address not accepted.
	MEM_WIDTH	0	8-bit FIFO data elements.
SCB_I2C_CTRL	MASTER_MODE	0	Disabled.
	SLAVE_MODE	1	Slave mode.
	S_GENERAL_IGNORE	0	General call Slave address enabled.

Note: Bits that are not listed in Table 24 have default values. For default values, see the respective registers in the Registers TRM.

Table 25 shows an example of the Rx configuration in the I²C Slave mode. These registers configure the receiver control settings. In this case, the received data format is MSb first and the data width is 8 bits. The SCB_RX_MATCH.ADDR register should be set to the Slave device address shifted left by 1 bit. The SCB_RX_MATCH.MASK register is set as is.

Table 25. I²C (Slave Mode): Example of Rx Configurations

Register	Bit	Value	Remark
SCB_RX_CTRL	MSB_FIRST	1	MSb first.
	DATA_WIDTH	7	DATA_WIDTH + 1 (=8) is the expected number of bits in the received data frame.
SCB_RX_FIFO_CTRL	TRIGGER_LEVEL	1	When the receiver FIFO has more entries than the number of this field, a receiver trigger event is generated.
	CLEAR	0	Not cleared.
	FREEZE	0	Not frozen.
SCB_RX_MATCH	ADDR	0x10	Slave device address =8. Set the Slave address shifted left by 1 bit because only bits 7 down to 1 (of 7 to 0) are used for the Slave address. In this case, Slave address: 1000b (=8) Set value: 0001 0000b (=0x10)
	MASK	0xFE	Slave device address mask. This is left-shifted by one bit.

Note: Bits that are not listed in Table 25 have default values. For default values, see the respective registers in the Registers TRM.

Table 26 shows an example of the Tx Configuration in I²C Slave mode. These registers configure the transmitter control settings. In this case, the transmitter data format is MSb first, the data width is 8bit, and OPEN_DRAIN mode is set.

Table 26. I²C (Slave Mode): Example of Tx Configurations

Register	Bit	Value	Remark
SCB_TX_CTRL	MSB_FIRST	1	MSb first.
	DATA_WIDTH	7	DATA_WIDTH + 1 (=8) is the number of bits in a transmitted data frame.
	OPEN_DRAIN	1	Open-drain operation mode.
SCB_TX_FIFO_CTRL	TRIGGER_LEVEL	64	When the transmitter FIFO has fewer entries than the number of this field, a transmitter trigger event is generated.

Note: Bits that are not listed in Table 26 have default values. For default values, see the respective registers in the Registers TRM.

Table 27 shows an example of the analog filter configuration in I²C Slave mode. These registers configure the analog filter settings to remove glitches. In this case, the filter of SDA_IN and SCLK_IN are enabled; SDA_OUT is disabled.

Table 27. I²C (Slave Mode): Example of Analog Filter Configurations

Register	Bit	Value	Remark
SCB_I2C_CFG	SDA_IN_FILT_SEL	1	SDA input filter is enabled.
	SCL_IN_FILT_SEL	1	SCL input filter is enabled.
	SDA_OUT_FILT_SEL	0	SDA output filter is disabled.

Note: Bits that are not listed in Table 27 have default values. For default values, see the respective registers in the Registers TRM.

Table 28 shows an example of the interrupt configuration in I²C Slave mode. These registers configure the interrupt mask control settings. In this case, arbitration lost, STOP event, Slave address match, I²C general call address receiving, and I²C bus error are set as receiver interrupts.

Table 28. I²C (Slave Mode): Example of Interrupt Configurations

Register	Bit	Value	Remark
SCB_INTR_I2C_EC_MASK	-	0	Externally clocked I²C interrupt request is masked.
SCB_INTR_M_MASK	-	0	Master interrupt is masked.
SCB_INTR_S_MASK	-	0x1D1	The following interrupt factor is enabled: I2C_ARB_LOST, I2C_STOP, I2C_ADDR_MATCH, I2C_GENERAL, I2C_BUS_ERROR.
SCB_INTR_TX_MASK	-	0	Transmitter interrupt is masked.
SCB_INTR_RX_MASK	-	0	Receiver interrupt is masked.

Note: Bits that are not listed in Table 28 have default values. For default values, see the respective registers in the Registers TRM.

7 Glossary

Terms	Description
SPI	Serial Peripheral Interface. SPI is a synchronous serial communication interface specification used for short distance communication with peripheral devices.
UART	Universal asynchronous receiver-transmitter. 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.
I^2C	Inter-Integrated Circuit. I^2C 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. I^2C bus is used with two lines of clock (SCK) and data (SDA) , and usually pulled-up by resistance.
Smart Card	Smart card is a card which integrated a circuit to record data and to operate it.
LIN	Local Interconnect Network. 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.
IrDA	IrDA is a kind of the standards of the optical radio data communication by infrared rays.
EZ mode	EZ (easy) mode is the Cypress original communication protocol which is prepared to simplify the Write/Read access between the device in SPI and I^2C. During DeepSleep mode, it can communicate with the Master device without CPU intervention.
CMD_RESP mode	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.
DMA	Direct Memory Access.
FIFO	First in First Out.

8 Related Documents

The following are the Traveo II 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 II Family
  - CYT2B9 Datasheet 32-Bit Arm® Cortex®-M4F Microcontroller Traveo II Family
  - CYT4BF Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo II Family
  - CYT4DN Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo II Family
  - CYT3BB/4BB Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo II Family
- Body Controller Entry Family
  - Traveo™ II Automotive Body Controller Entry Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B7
  - Traveo™ II Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B9
- Body Controller High Family
  - Traveo™ II Automotive Body Controller High Family Architecture Technical Reference Manual (TRM) for CYT4BF
  - Traveo™ II Automotive Body Controller High Registers Technical Reference Manual (TRM) for CYT3BB/4BB
- Cluster 2D Family
  - Traveo™ II Automotive Cluster 2D Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Cluster 2D Registers Technical Reference Manual (TRM)

Document History

Document Title: AN225401 - How to Use Serial Communications Block (SCB) in Traveo II Family

Document Number: 002-25401

Revision	ECN	Submission Date	Description of Change
**	6428773	07/09/2019	New application note.
*A	6736694	11/22/2019	Added a part number CYT4D series.
*B	6806984	03/02/2020	Added parts number CYT2/CYT3/CYT4 series.

Worldwide Sales and Design Support

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

Products

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

PSoC® Solutions

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

Cypress Developer Community

Community | Code Examples | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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