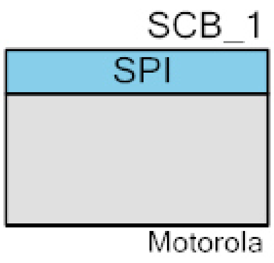
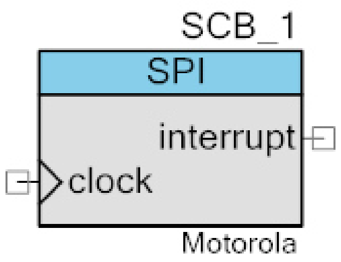
SPI





This component provides an industry-standard, 4-wire SPI interface. Three different SPI protocols or modes are supported:

- Original SPI protocol as defined by Motorola.
- TI: Uses a short pulse on “spi_select” to indicate start of transaction.
- National Semiconductor (Microwire): Transmission and Receptions occur separately.

In addition to the standard 8-bit word length, the component supports a configurable 4 to 16-bit data width for communicating at nonstandard SPI data widths.

Input/Output Connections

This section describes the various input and output connections for the SCB component. An asterisk (*) in the list of terminals indicates that the terminal may be hidden on the symbol under the conditions listed in the description of that terminal.

clock – Input*

Clock that operates this block. The presence of this terminal varies depending on the **Clock from terminal** parameter.

interrupt – Output*

This signal can only be connected to an interrupt component or left unconnected. The presence of this terminal varies depending on the **Interrupt** parameter.

rx_tr_out – Output*

This signal can only be connected to a DMA channel trigger input. This signal is used to trigger a DMA transaction. The output of this terminal is controlled by the RX FIFO level. The presence of this terminal varies depending **RX Output** parameter.

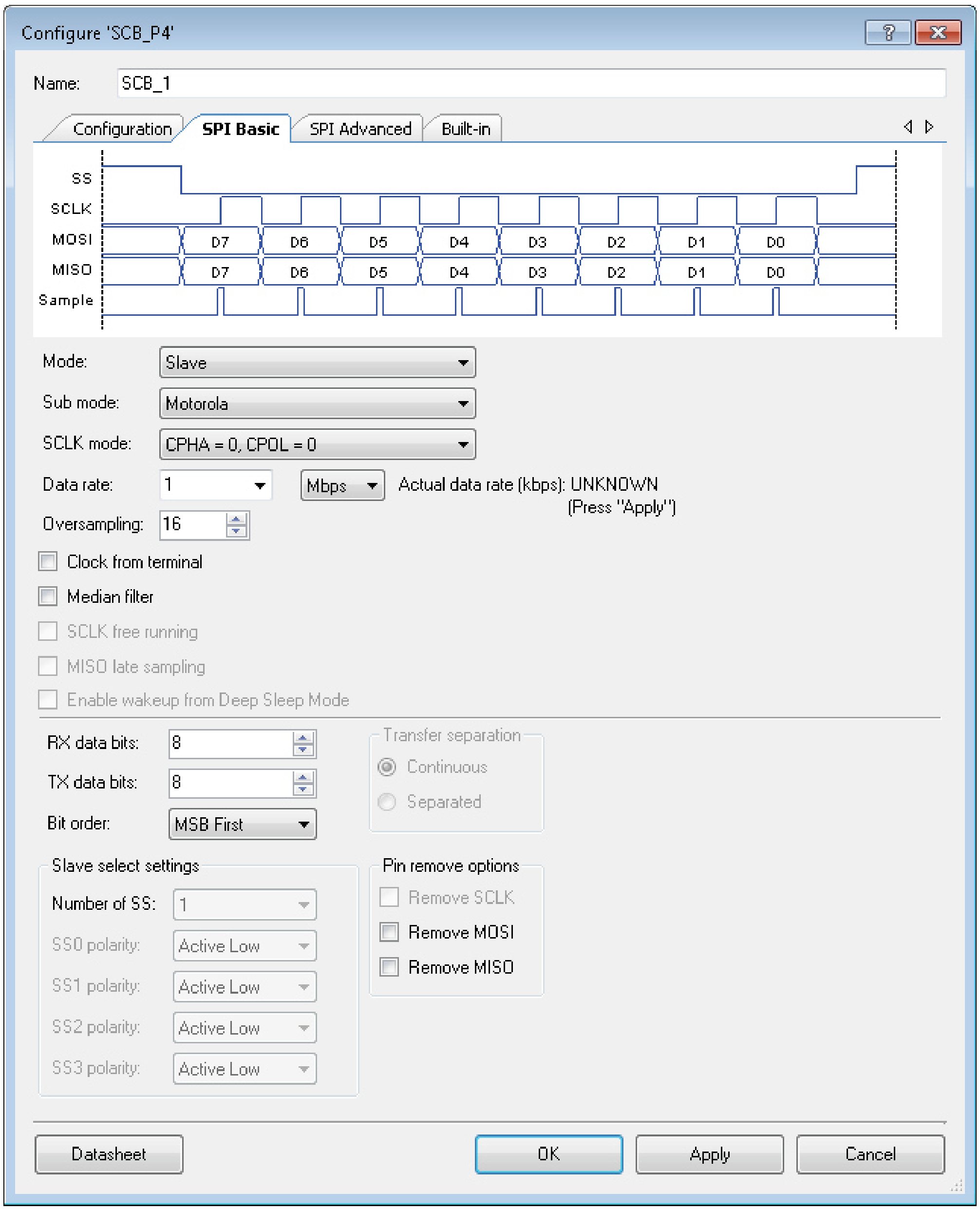
tx_tr_out – Output*

This signal can only be connected to a DMA channel trigger input. This signal is used to trigger a DMA transaction. The output of this terminal is controlled by the TX FIFO level. The presence of this terminal varies depending **TX Output** parameter.

The interface-specific pins are buried inside component because these pins use dedicated connections and are not routable as general purpose signals. See the *I/O System* section in the device *Technical Reference Manual (TRM)* for more information.

Note The input buffer of buried output pins is disabled so as not to cause current linkage in low power mode. Reading the status of these pins always returns zero. To get the current status, the input buffer must be enabled before status read.

Basic SPI Parameters



Name: SCB_1

Mode: Slave

Sub mode: Motorola

SCLK mode: CPHA = 0, CPOL = 0

Data rate: 1 Mbps Actual data rate (kbps): UNKNOWN (Press "Apply")

Oversampling: 16

☐ Clock from terminal
☐ Median filter
☐ SCLK free running
☐ MISO late sampling
☐ Enable wakeup from Deep Sleep Mode

RX data bits: 8

TX data bits: 8

Bit order: MSB First

Transfer separation: ● Continuous ○ Separated

Number of SS: 1

SS0 polarity: Active Low

SS1 polarity: Active Low

SS2 polarity: Active Low

SS3 polarity: Active Low

☐ Remove SCLK
☐ Remove MOSI
☐ Remove MISO

The **SPI Basic** tab contains the following parameters:

Mode

This option determines in which SPI mode the SCB operates.

- **Slave** – Slave only operation (default)
- **Master** – Master only operation

Sub mode

This option determines what SPI sub-modes are supported:

- **Motorola** – The original SPI protocol as defined by Motorola (default).
- **TI (Start Coincides)** – The Texas Instruments' SPI protocol.
- **TI (Start Precedes)** – The Texas Instruments' SPI protocol.
- **National Semiconductor (Microwire)** – The National Semiconductor's Microwire protocol.

SCLK mode

This parameter defines the serial clock phase and clock polarity mode for communication.

- **CPHA = 0, CPOL= 0 –** Data is driven on a falling edge of SCLK. Data is captured on a rising edge of SCLK. SCLK idles low.This is default mode.
- **CPHA = 0, CPOL= 1** – Data is driven on a rising edge of SCLK. Data is captured on a falling edge of SCLK. SCLK idles high.
- **CPHA = 1, CPOL= 0** – Data is driven on a rising edge of SCLK. Data is captured on a falling edge of SCLK. SCLK idles low
- **CPHA = 1, CPOL= 1** – Data is driven on a falling edge of SCLK. Data is captured on a rising edge of SCLK. SCLK idles high

Refer to the section Motorola sub mode operation in the SPI chapter of this document for more information.

Data rate

This parameter is used to set the SPI data rate value up to 8000 kbps; the actual rate may differ based on available clock frequency and component settings. The standard data rates are 500, 1000 (default), 2000, 4000 to 8000 in multiples of 2000 kbps. This parameter has no effect if the **Clock from terminal** parameter is enabled.

Actual data rate

The actual data rate displays the data rate at which the component will operate with current settings. The factors that affect the actual data rate calculation are: the accuracy of the component clock (internal or external) and oversampling factor (only for the Master mode). When a change is made to any of the component parameters that affect actual data rate, it becomes unknown. To calculate the new actual data rate press the Apply button.

Note For Slave mode the actual data rate always provides maximum value for the selected clock frequency. As external Master parameters are unknown, the assumption was made that MISO is sampled in the leading edge of SCLK.

Refer to the Slave data rate section for actual data rate calculation which takes to account external environment timing conditions.

Oversampling

This parameter defines the oversampling factor of the SPI clock; the number of component clocks within one SPI clock period. Oversampling factor is used to calculate the internal component clock frequency required to achieve this amount of oversampling as follows: SCBCLK = Data rate * Oversampling factor.

- For **Slave** mode, only the component clock source frequency is important. The oversampling value is used to create a clock fast enough to operate at the selected data rate. Refer to the Maximum data rate calculation section for more information. The created clock is equal to the (data rate * Oversampling).
- For **Master** mode, the oversampling value is used for serial clock signal (SCLK) generation. The oversampling is equal to number of component clocks within one SPI clock period. When the oversampling is even the first and second phase of the clock period are the same. Otherwise the first phase of the clock signal period is one component clock cycle longer than the second phase. The level of the first phase of the clock period depends on CPOL settings: 0 – low level and 1 – high level.

An oversampling factor maximum value is 16 and minimum depends on component settings. For **Master** the minimum oversampling factor value is 6. For **Slave** the minimum oversampling value 6 (Median filter is disabled) or 8 (Median filter is enabled).

Clock from terminal

This parameter allows choosing between an internally configured clock (by the component) or an externally configured clock (by the user) for the component operation. Refer to the **Oversampling** section to understand relationship between component clock frequency and the component parameters.

When this option is enabled, the component does not control the data rate, but displays the actual data rate based on the user-connected clock source frequency and the component oversampling factor (only for the Master mode). When this option is not enabled, the clock configuration and Oversampling factor (only for the Master mode) is provided by the component. The clock source frequency is calculated by the component based on the Data rate parameter.

Note PSoC Creator is responsible for providing requested clock frequency (internal or external clock) based on current design clock configuration. When the requested clock frequency with requested tolerance cannot be created, a warning about the clock accuracy range is generated while building the project. This warning contains the actual clock frequency value created by

PSoC Creator. To remove this warning you must either change the system clock, component settings or external clock to fit the clocking system requirements.

Median filter

This parameter applies 3 taps digital median filter on the input line. The master has one input line: MISO, and the slave has three input lines: SCLK, MOSI, and SS. This filter reduces the susceptibility to errors. However, minimum oversampling factor value is increased. The default value is a **Disabled**.

SCLK free running

This option is only applicable for PSoC 4100 BLE/PSoC 4200 BLE/PSoC 4100M/PSoC 4200M/ PSoC 4200L devices in Master mode. It allows master to generate SCLK continually. It is useful when master SCLK is connected to the slave device which uses it for functional operation rather than just SPI functionality.

The default value is a **Disabled**.

Enable late MISO sample

This option allows the master to sample the MISO signal by half of SCLK period later (on the alternate serial clock edge). Late sampling addresses the round-trip delay associated with transmitting SCLK from the master to the slave and transmitting MISO from the slave to the master. The default value is a **Disabled**.

Enable wakeup from Deep Sleep Mode

Use this option to enable the component to wake the system from Deep Sleep when slave select occurs.

To enable this option all of the following restrictions must be met:

- Sub mode is Motorola
- SCLK mode is CPHA = 0, CPOL = 0 (only for PSoC 4100/PSoC 4200 devices)
- Interrupt is Internal

Refer to the Low power modes section under the SPI chapter in this document and *Power Management APIs* section of the *System Reference Guide* for more information.

TX data bits

This option defines the bit width in a transmitted data frame. The default number of bits is a single byte (8 bits). Any integer from 4 to 16 is a valid setting.

RX data bits

This option defines the bit width in a received data frame. The default number of bits is a single byte (8 bits). Any integer from 4 to 16 is a valid setting.

Note The number of **TX data bits** and **RX data bits** should be set the same for **Motorola** and **Texas Instruments** sub-modes; they can be set different for **National Semiconductor** sub-mode.

Bit order

The **Bits order** parameter defines the direction in which the serial data is transmitted. When set to **MSB first**, the most-significant bit is transmitted first. When set to **LSB first**, the least-significant bit is transmitted first.

Remove SCLK

This option allows removal of the SCLK pin from the SPI interface. If selected, this pin is no longer available in the Design-Wide Resources System Editor (in the *<project>.cydwr* file) on the Pins Tab. The SCLK pin cannot be removed in Slave mode.

Remove MOSI

This option allows removal of the MOSI pin from the SPI interface. If selected, this pin is no longer available in the Design-Wide Resources System Editor (in the *<project>.cydwr* file) on the Pins Tab.

Remove MISO

This option allows removal of the MISO pin from the SPI interface. If selected, this pin is no longer available in the Design-Wide Resources System Editor (in the *<project>.cydwr* file) on the Pins Tab.

Number of SS

This parameter determines the number of SPI slave select lines. Only one slave select line is available in Slave mode and it is not optional. The values between 0 and 4 are valid choices in Master mode. The default number of lines is 1.

Transfer separation

This parameter determines if individual data transfers are separated by slave select de-selection (only applicable for Master mode):

- **Continuous –** The slave select line is held in active state until the end of transfer (default).

  The master assigns the slave select output after data has been written into the TX FIFO and keeps it active as long as there are data elements to transmit. The slave select output becomes inactive when all data elements have been transmitted from the TX FIFO and shifter register.

Note This can happen even in the middle of the transfer if the TX FIFO is not loaded fast enough by the CPU or DMA. To overcome this behavior, the slave select can be controlled by firmware. For more information about slave select control, refer to the Slave select lines section.

- **Separated –** Every data frame 4-16 bits is separated by slave select line de-selection by one SCLK period.

SS0-SS3 polarity

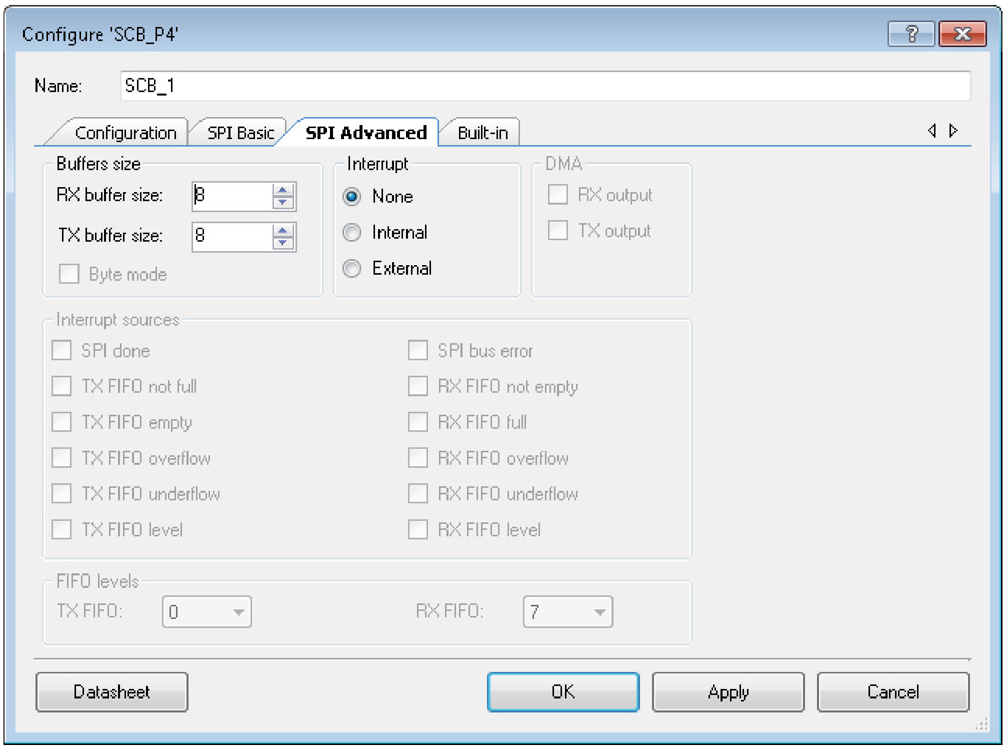
This option is only applicable for PSoC 4100 BLE/PSoC 4200 BLE/PSoC 4100M/PSoC 4200M/ PSoC 4200L devices. It determines the active polarity of the slave select signal as Active Low (default) or Active High. For other devices, only Active Low is available.

Each slave select line active polarity can be configured independently.

For Texas Instruments precede/coincide sub-modes the active polarity logic is inverted:

- **Active Low –** Slave select line is inactive low and generated pulse is active high.
- **Active High –** Slave select line is inactive high and generated pulse is active low.

Advanced SPI Parameters



Name: SCB_1

Buffers size
RX buffer size: 8
TX buffer size: 8
☐ Byte mode

Interrupt
[✓ None] Internal External

DMA
☐ RX output
☐ TX output

Interrupt sources
☐ SPI done
☐ TX FIFO not full
☐ TX FIFO empty
☐ TX FIFO overflow
☐ TX FIFO underflow
☐ TX FIFO level
☐ SPI bus error
☐ RX FIFO not empty
☐ RX FIFO full
☐ RX FIFO overflow
☐ RX FIFO underflow
☐ RX FIFO level

FIFO levels
TX FIFO: 0
RX FIFO: 7

The **SPI Advanced** tab contains the following parameters:

RX buffer size

The **RX buffer size** parameter defines the size (in bytes/words) of memory allocated for a receive data buffer. The minimum value is equal to the RX FIFO depth. The RX FIFO is implemented in hardware. Values greater than the RX FIFO depth up to (2^{32} – 2) imply using the RX FIFO, and a circular software buffer controlled by the supplied APIs, and the internal interrupt handler. The software buffer size is limited only by the available memory. The interrupt mode is automatically set to internal and the RX FIFO not empty interrupt source is reserved if a software buffer is used.

- For 4100/PSoC 4200 devices, the RX and TX FIFO depth is equal to 8 bytes/words.
- For PSoC 4100 BLE/PSoC 4200 BLE/PSoC 4100M/PSoC 4200M/PSoC 4200L devices, the RX and TX FIFO depth is equal to 8 bytes/words or 16 bytes; refer to **Byte mode** for more information.

TX buffer size

The **TX buffer size** parameter defines the size (in bytes/words) of memory allocated for a circular transmit data buffer. The TX buffer size minimum value is equal to the TX FIFO depth. The TX FIFO is implemented in hardware. Values greater than the TX FIFO depth up to (2^{32} – 1) imply using the TX FIFO, circular software buffer controlled by the supplied APIs, and the internal interrupt handler. The software buffer size is limited only by the available memory. The interrupt mode is automatically set to the internal and the TX FIFO not full interrupt source is reserved if a software buffer is used.

- For 4100/PSoC 4200 devices, the RX and TX FIFO depth is equal to 8 bytes/words.
- For PSoC 4100 BLE/PSoC 4200 BLE/PSoC 4100M/PSoC 4200M/PSoC 4200L devices, the RX and TX FIFO depth is equal to 8 bytes/words or 16 bytes; refer to **Byte mode** for more information.

Byte mode

This option is only applicable to PSoC 4100 BLE/PSoC 4200 BLE/PSoC 4100M/PSoC 4200M/PSoC 4200L devices. It allows doubling the TX and RX FIFO depth from 8 to 16 bytes, by reducing the FIFO width from 16bits to 8 bits. This implies that the number of TX and RX data bits must be less than or equal to 8 bits. Increasing FIFO depth improves performance of SPI operation as more bytes can be transmitted or received without software interaction.

Interrupt

This option determines what interrupt modes are supported None, Internal or External.

- **None** – This option removes the internal interrupt component.

- **Internal** – This option leaves the interrupt component inside the SCB component. The predefined internal interrupt handler is hooked up to the interrupt. The **Interrupt sources** option sets one or more interrupt sources, which trigger the interrupt. To add your own code to the interrupt service routine you need to register a function using the SCB_SetCustomInterruptHandler() function.
- **External** – This option removes the internal interrupt and provides an output terminal. Only an interrupt component can be connected to this terminal if an interrupt handler is desired. The **Interrupt sources** option sets one or more interrupt sources, which trigger the interrupt output.

Note For buffer sizes greater than the hardware FIFO depth, the component automatically enables the internal interrupt sources required for proper internal software buffer operations. In addition, the global interrupt enable must be explicitly enabled for proper buffer handling.

DMA

DMA is only available in PSoC 4100M/PSoC 4200M/PSoC 4200L devices. The **RX Output** and **TX Output** options determine if DMA output trigger terminals are available on the component symbol.

RX Output

This option determines if the rx_tr_out terminal is available on the component symbol. This signal can only be connected to a DMA channel trigger input. The output of this terminal is controlled by the RX FIFO level. This option is active only when RX buffer size equal to FIFO depth.

TX Output

This option determines if the tx_tr_out terminal is available on the component symbol. This signal can only be connected to a DMA channel trigger input. The output of this terminal is controlled by the TX FIFO level. This option is active only when TX buffer size equal to FIFO depth.

Interrupt sources

The interrupt sources are either level or pulse. Level interrupt sources in the following list are indicated with an asterisk (*). Refer to sections TX FIFO interrupt sources and RX FIFO interrupt sources for more information about level interrupt sources operation. The SPI supports interrupts on the following events:

- **SPI done** – Master transfer done event: all data elements from the TX FIFO are sent. This interrupt source triggers later than TX FIFO empty by the amount of time it takes to transmit a single data element. The TX FIFO empty triggers when the last data element from the TX FIFO goes to the shifter register. However, SPI done triggers after this data element has been transmitted. This means SPI done will be asserted one SCLK clock cycle earlier than the reception of the data element has been completed. It is recommended to use SCB_SpiIsBusBusy() after checking SPI done to determine when the data element reception has been fully completed. As an alternative, the number of received data elements can be checked to make sure that it is equal to the number of the transmitted data elements.

- **TX FIFO not full * –** TX FIFO is not full. At least one data element can be written into the TX FIFO.
- **TX FIFO empty * –** TX FIFO is empty.
- **TX FIFO overflow –** Firmware attempts to write to a full TX FIFO.
- **TX FIFO underflow –** Hardware attempts to read from an empty TX FIFO.
- **TX FIFO level * –** An interrupt request is generated whenever the number of data elements in the TX FIFO is less than the value of TX FIFO level.
- **SPI bus error –** SPI slave deselected at an unexpected time during the SPI transfer.
- **RX FIFO not empty * –** RX FIFO is not empty. At least one data element is available in the RX FIFO to be read.
- **RX FIFO full * –** RX FIFO is full.
- **RX FIFO overflow –** Hardware attempts to write to a full RX FIFO.
- **RX FIFO underflow –** Firmware attempts to read from an empty RX FIFO.
- **RX FIFO level * –** An interrupt request is generated whenever the number of data elements in the RX FIFO is greater than the value of RX FIFO level.

Notes

When **RX buffer size** is greater than the RX FIFO depth, the **RX FIFO not empty** interrupt source is reserved by the component and used for the internal interrupt.

When **TX buffer size** is greater than the TX FIFO depth, the **TX FIFO not full** interrupt source is reserved by the component and used for the internal interrupt.

FIFO level

The RX and TX FIFO level settings control behavior of the appropriate level interrupt sources as well as RX and TX DMA triggers outputs.

RX FIFO

The interrupt or DMA trigger output signal remains active until the number of data elements in the RX FIFO is greater than the value of RX FIFO level.

TX FIFO

The interrupt or DMA trigger output signal remains active until the number of data elements in the TX FIFO is less than the value of TX FIFO level.

SPI APIs

APIs allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections discuss each function in more detail.

By default, PSoC Creator assigns the instance name "SCB_1" to the first instance of a component in a given design. You can rename the instance to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "SCB".

SCB_Start()	Starts the SCB.
SCB_Init()	Initialize the SCB component according to defined parameters in the customizer.
SCB_Enable()	Enables SCB component operation.
SCB_Stop()	Disable the SCB component.
SCB_Sleep()	Prepares component to enter Deep Sleep.
SCB_Wakeup()	Prepares component for Active mode operation after Deep Sleep.
SCB_SpiInit()	Configures the SCB for SPI operation.
SCB_SpiIsBusBusy()	Returns the current status on the bus.
SCB_SpiSetActiveSlaveSelect()	Selects the active slave select line. Only applicable in Master mode.
SCB_SpiSetSlaveSelectPolarity()	Sets active polarity for the slave select line.
SCB_SpiUartWriteTxData()	Places a data entry into the transmit buffer to be sent at the next available bus time.
SCB_SpiUartPutArray()	Places an array of data into the transmit buffer to be sent.
SCB_SpiUartGetTxBufferSize()	Returns the number of elements currently in the transmit buffer.
SCB_SpiUartClearTxBuffer()	Clears the transmit buffer and TX FIFO.
SCB_SpiUartReadRxData()	Retrieves the next data element from the receive buffer.
SCB_SpiUartGetRxBufferSize()	Returns the number of received data elements in the receive buffer.
SCB_SpiUartClearRxBuffer()	Clears the receive buffer and RX FIFO.

void SCB_Start(void)

Description:	Invokes SCB_Init() and SCB_Enable(). After this function call the component is enabled and ready for operation. This is the preferred method to begin component operation. When configuration is set to "Unconfigured SCB", the component must first be initialized to operate in one of the following configurations: I^2C, SPI, UART or EZ I^2C. Otherwise this function does not enable component.
Parameters:	None
Return Value:	None
Side Effects:	None

void SCB_Init(void)

Description:	Initializes the SCB component to operate in one of the selected configurations: I^2C, SPI, UART or EZ I^2C. When configuration is set to "Unconfigured SCB", this function does not do any initialization. Use mode-specific initialization APIs instead: SCB_I2CInit, SCB_SpiInit, SCB_UartInit or SCB_EzI2CInit.
Parameters:	None
Return Value:	None
Side Effects:	None

void SCB_Enable(void)

Description:	Enables SCB component operation: activates the hardware and internal interrupt. For I^2C and EZ I^2C modes the interrupt is internal and mandatory for operation. For SPI and UART modes the interrupt can be configured as none, internal or external. The SCB configuration should be not changed when the component is enabled. Any configuration changes should be made after disabling the component. When configuration is set to "Unconfigured SCB", the component must first be initialized to operate in one of the following configurations: I^2C, SPI, UART or EZ I^2C. Otherwise this function does not enable component.
Parameters:	None
Return Value:	None
Side Effects:	None

void SCB_Stop(void)

Description: Disables the SCB component: disable the hardware and internal interrupt. It also disables all TX interrupt sources so as not to cause an unexpected interrupt trigger because after the component is enabled, the TX FIFO is empty.

Refer to the function SCB_Enable() for the interrupt configuration details.

This function disables the SCB component without checking to see if communication is in progress. Before calling this function it may be necessary to check the status of communication to make sure communication is complete. If this is not done then communication could be stopped mid byte and corrupted data could result.

Parameters: None

Return Value: None

Side Effects: None

void SCB_Sleep(void)

Description: Prepares component to enter Deep Sleep.

The “Enable wakeup from Deep Sleep Mode” selection has an influence on this function implementation:

- Checked: configures the component to be wakeup source from Deep Sleep.
- Unchecked: stores the current component state (enabled or disabled) and disables the component. See SCB_Stop() function for details about component disabling.

Call the SCB_Sleep() function before calling the CyPmSysDeepSleep() function. Refer to the PSoC Creator *System Reference Guide* for more information about power-management functions.

This function should not be called before entering Sleep.

Parameters: None

Return Value: None

Side Effects: None

void SCB_Wakeup(void)

Description:	Prepares component for Active mode operation after Deep Sleep. The “Enable wakeup from Deep Sleep Mode” selection has influence to on this function implementation: • Checked: restores the component Active mode configuration. • Unchecked: enables the component if it was enabled before enter Deep Sleep. **This function should not be called after exiting Sleep.**
Parameters:	None
Return Value:	None
Side Effects:	Calling the SCB_Wakeup() function without first calling the SCB_Sleep() function may produce unexpected behavior.

void SCB_SpiInit(SCB_SPI_INIT_STRUCT *config)

Description: Configures the SCB for SPI operation.

This function is **intended specifically** to be used when the SCB configuration is set to “Unconfigured SCB” in the customizer. After initializing the SCB in SPI mode, the component can be enabled using the SCB_Start() or SCB_Enable() function.

This function uses a pointer to a structure that provides the configuration settings. This structure contains the same information that would otherwise be provided by the customizer settings.

Parameters: config: pointer to a structure that contains the following list of fields. These fields match the selections available in the customizer. Refer to the customizer for further description of the settings.

uint32 mode	Mode of operation for SPI. The following defines are available choices: SCB_SPI_SLAVE SCB_SPI_MASTER
uint32 submode	Submode of operation for SPI. The following defines are available choices: SCB_SPI_MODE_MOTOROLA SCB_SPI_MODE_TI_COINCIDES SCB_SPI_MODE_TI_PRECEDES SCB_SPI_MODE_NATIONAL
uint32 sclkMode	Determines the sclk relationship for Motorola submode. Ignored for other submodes. The following defines are available choices: SCB_SPI_SCLK_CPHA0_CPOL0 SCB_SPI_SCLK_CPHA0_CPOL1 SCB_SPI_SCLK_CPHA1_CPOL0 SCB_SPI_SCLK_CPHA1_CPOL1
uint32 oversample	Oversampling factor for the SPI clock. Ignored for Slave mode operation.
uint32 enableMedianFilter	0 – disable 1 – enable
uint32 enableLateSampling	0 – disable 1 – enable Ignored for slave mode.
uint32 enableWake	0 – disable 1 – enable Ignored for master mode.
uint32 rxDataBits	Number of data bits for RX direction. Different dataBitsRx and dataBitsTx are only allowed for National submode.
uint32 txDataBits	Number of data bits for TX direction. Different dataBitsRx and dataBitsTx are only allowed for National submode.

uint32 bitOrder	Determines the bit ordering. The following defines are available choices: SCB_BITS_ORDER_LSB_FIRST SCB_BITS_ORDER_MSB_FIRST
uint32 transferSeperation	Determines whether transfers are back to back or have SS disabled between words. Ignored for slave mode. The following defines are available choices: SCB_SPI_TRANSFER_CONTINUOUS SCB_SPI_TRANSFER_SEPARATED
uint32 rxBufferSize	Size of the RX buffer in bytes/words (depends on rxDataBits parameter). A value equal to the RX FIFO depth implies the usage of buffering in hardware. A value greater than the RX FIFO depth results in a software buffer. The SCB_INTR _RX_NOT_EMPTY interrupt has to be enabled to transfer data into the software buffer. For 4100/PSoC 4200 devices, the RX and TX FIFO depth is equal to 8 bytes/words. For PSoC 4100 BLE/PSoC 4200 BLE/PSoC 4100M/PSoC 4200M/PSoC 4200L devices, the RX and TX FIFO depth is equal to 8 bytes/words or 16 bytes (Byte mode is enabled).
uint8* rxBuffer	Buffer space provided for a RX software buffer: • A NULL pointer must be provided to use hardware buffering. • A pointer to an allocated buffer must be provided to use software buffering. The buffer size must equal (rxBufferSize + 1) in bytes if dataBitsRx is less or equal to 8, otherwise (2 * (rxBufferSize + 1)) in bytes. The software RX buffer always keeps one element empty. For correct operation the allocated RX buffer has to be one element greater than maximum packet size expected to be received.
uint32 txBufferSize	Size of the TX buffer in bytes/words(depends on txDataBits parameter). A value equal to the TX FIFO depth implies the usage of buffering in hardware. A value greater than the TX FIFO depth results in a software buffer. For 4100/PSoC 4200 devices, the RX and TX FIFO depth is equal to 8 bytes/words. For PSoC 4100 BLE/PSoC 4200 BLE/PSoC 4100M/PSoC 4200M/PSoC 4200L devices, the RX and TX FIFO depth is equal to 8 bytes/words or 16 bytes (Byte mode is enabled).
uint8* txBuffer	Buffer space provided for a TX software buffer: • A NULL pointer must be provided to use hardware buffering. • A pointer to an allocated buffer must be provided to use software buffering. The buffer size must equal txBufferSize if dataBitsTx is less or eqal to 8, otherwise (2* rxBufferSize).
uint32 enableInterrupt	0 – disable 1 – enable The interrupt has to be enabled if software buffer is used.

uint32 rxInterruptMask	Mask of enabled interrupt sources for the RX direction. This mask is written regardless of the setting of the enable Interrupt field. Multiple sources are enabled by providing a value that is the OR of all of the following sources to enable: • SCB_INTR_RX_FIFO_LEVEL • SCB_INTR_RX_NOT_EMPTY • SCB_INTR_RX_FULL • SCB_INTR_RX_OVERFLOW • SCB_INTR_RX_UNDERFLOW • SCB_INTR_SLAVE_SPI_BUS_ERROR
uint32 rxTriggerLevel	FIFO level for an RX FIFO level interrupt. This value is written regardless of whether the RX FIFO level interrupt source is enabled.
uint32 txInterruptMask	Mask of enabled interrupt sources for the TX direction. This mask is written regardless of the setting of the enable Interrupt field. Multiple sources are enabled by providing a value that is the OR of all of the following sources to enable: • SCB_INTR_TX_FIFO_LEVEL • SCB_INTR_TX_NOT_FULL • SCB_INTR_TX_EMPTY • SCB_INTR_TX_OVERFLOW • SCB_INTR_TX_UNDERFLOW • SCB_INTR_MASTER_SPI_DONE
uint32 txTriggerLevel	FIFO level for a TX FIFO level interrupt. This value is written regardless of whether the TX FIFO level interrupt source is enabled.
uint8 enableByteMode	Ignored for all devices other than For PSoC 4100 BLE/ PSoC 4200 BLE/ PSoC 4100M/PSoC 4200M/PSoC 4200L. 0 – disable 1 – enable When enabled the TX and RX FIFO depth is 16 bytes. This implies that number of TX and RX data bits must be less than or equal to 8.
uint8 enableFreeRunSclk	Ignored for all devices other than For PSoC 4100 BLE/ PSoC 4200 BLE/PSoC 4100M/PSoC 4200M/PSoC 4200L. Enables continuous SCLK generation by the SPI master. 0 – disable 1 – enable
uint8 polaritySs	Ignored for all devices other than For PSoC 4100 BLE/ PSoC 4200 BLE/PSoC 4100M/PSoC 4200M/PSoC 4200L. Active polarity of slave select lines 0-3. This is bitmask where bit SCB_SPI_SLAVE_SELECT0 corresponds to slave select 0 polarity, bit SCB_SPI_SLAVE_SELECT1 – slave select 1 polarity and so on. Polarity constants are: SCB_SPI_SS_ACTIVE_LOW SCB_SPI_SS_ACTIVE_HIGH

Return Value: None

Side Effects: None

uint32 SCB_SpiIsBusBusy(void)

Description: Returns the current status on the bus. The bus status is determined using the slave select signal.

- Motorola and National Semiconductor sub-modes: The bus is busy after the slave select line is activated and lasts until the slave select line is deactivated.
- Texas Instrument sub-modes: The bus is busy at the moment of the initial pulse on the slave select line and lasts until the transfer is complete.

If SPI Master is configured to use "separated transfers" (see Continuous versus Separated Transfer Separation), the bus is busy during each element transfer and is free between each element transfer. The Master does not activate SS line immediately after data has been written into the TX FIFO.

Parameters: None

Return Value: uint32: Current status on the bus. If the returned value is nonzero, the bus is busy. If zero is returned, the bus is free. The bus status is determined using the slave select signal.

Side Effects: None

void SCB_SpiSetActiveSlaveSelect(uint32 slaveSelect)

Description: Selects one of the four slave select lines to be active during the transfer. After initialization the active slave select line is 0.

The component should be in one of the following states to change the active slave select signal source correctly:

- The component is disabled
- The component has completed transfer

This function does not check that these conditions are met.

This function is only applicable to SPI Master mode of operation.

Parameters: uint32 slaveSelect: slave select line that will be active after the transfer.

SCB_SPI_SLAVE_SELECT0	Slave select 0
SCB_SPI_SLAVE_SELECT1	Slave select 1
SCB_SPI_SLAVE_SELECT2	Slave select 2
SCB_SPI_SLAVE_SELECT3	Slave select 3

Return Value: None

Side Effects: None

void SCB_SpiSetSlaveSelectPolarity(uint32 slaveSelect, uint32 polarity)

Description: Sets active polarity for the slave select line.

The component should be in one of the following states to change the active slave select signal correctly:

- The component is disabled
- The component has completed transfer

This function does not check that these conditions are met.

Parameters: uint32 slaveSelect: slave select line to change active polarity.

SCB_SPI_SLAVE_SELECT0	Slave select 0. For SPI slave mode the slave select 0 is only valid argument due to slave select placement constraint.
SCB_SPI_SLAVE_SELECT1	Slave select 1
SCB_SPI_SLAVE_SELECT2	Slave select 2
SCB_SPI_SLAVE_SELECT3	Slave select 3

uint32 Polarity: active polarity of slave select line.

SCB_SPI_SS_ACTIVE_LOW	Slave select is active low
SCB_SPI_SS_ACTIVE_HIGH	Slave select is active high

Return Value: None

Side Effects: None

void SCB_SpiUartWriteTxData(uint32 txData)

Description: Places a data entry into the transmit buffer to be sent at the next available bus time.

This function is blocking and waits until there is space available to put the requested data in the transmit buffer.

Parameters: uint32 txData: the data to be transmitted.

The amount of data bits to be transmitted depends on TX data bits selection (the data bit counting starts from LSB of txDataByte).

Return Value: None

Side Effects: None

void SCB_SpiUartPutArray(const uint16/uint8 wrBuf[], uint32 count)

Description:	Places an array of data into the transmit buffer to be sent. This function is blocking and waits until there is a space available to put all the requested data in the transmit buffer. The array size can be greater than transmit buffer size.
Parameters:	const uint16/uint8 wrBuf[]: pointer to an array of data to be placed in transmit buffer. The width of the data to be transmitted depends on TX data width selection (the data bit counting starts from LSB for each array element). uint32 count: number of data elements to be placed in the transmit buffer.
Return Value:	None
Side Effects:	None

uint32 SCB_SpiUartGetTxBufferSize(void)

Description:	Returns the number of elements currently in the transmit buffer. <u>TX software buffer is disabled:</u> Returns the number of used entries in TX FIFO. <u>TX software buffer is enabled:</u> Returns the number of elements currently used in the transmit buffer. This number does not include used entries in the TX FIFO. The transmit buffer size is zero until the TX FIFO is not full.
Parameters:	None
Return Value:	uint32: Number of data elements ready to transmit.
Side Effects:	None

void SCB_SpiUartClearTxBuffer(void)

Description:	Clears the transmit buffer and TX FIFO.
Parameters:	None
Return Value:	None
Side Effects:	None

uint32 SCB_SpiUartReadRxData(void)

Description:	Retrieves the next data element from the receive buffer. RX software buffer is disabled: Returns data element retrieved from RX FIFO. Undefined data will be returned if the RX FIFO is empty. RX software buffer is enabled: Returns data element from the software receive buffer. Zero value is returned if the software receive buffer is empty.
Parameters:	None
Return Value:	uint32: Next data element from the receive buffer. The amount of data bits to be received depends on RX data bits selection (the data bit counting starts from LSB of return value).
Side Effects:	None

uint32 SCB_SpiUartGetRxBufferSize(void)

Description:	Returns the number of received data elements in the receive buffer. RX software buffer is disabled: Returns the number of used entries in RX FIFO. RX software buffer is enabled: Returns the number of elements that were placed in the receive buffer. This does not include the hardware RX FIFO.
Parameters:	None
Return Value:	uint32: Number of received data elements
Side Effects:	None

void SCB_SpiUartClearRxBuffer(void)

Description:	Clears the receive buffer and RX FIFO.
Parameters:	None
Return Value:	None
Side Effects:	None

Global Variables

Knowledge of these variables is not required for normal operations.

SCB_initVar	SCB_initVar indicates whether the SCB component has been initialized. The variable is initialized to 0 and set to 1 the first time SCB_Start() is called. This allows the component to restart without reinitialization after the first call to the SCB_Start() routine. If reinitialization of the component is required, then the SCB_Init() function can be called before the SCB_Start() or SCB_Enable() function.

SCB_rxBufferOverflow	SCB_rxBufferOverflow sets when internal software receive buffer overflow was occurred.

Bootloader Support

The SCB component in SPI mode can be used as a communication component for the Bootloader. You should use the following configuration to support the SPI communication protocol from an external system to the Bootloader:

- SPI Mode: Slave
- Sub Mode: Motorola
- Data Lines: MOSI, MISO, SCLK, SS
- TX data bits and RX data bits: 8
- SCLK mode: Must match Host (boot device)
- Data rate: Must not be less than Host (boot device)

  Note The slave uses input signal oversampling to allow the master to successfully communicate with the slave at a data rate lower than what is selected in the Configure dialog. However, when the component is used for the bootloading, the selected data rate is used to calculate the byte-to-byte timeout interval. This timeout interval can be too small if the data rate, used by the master to communicate with the slave, is significantly lower than the data rate set in the Configure dialog. If the timeout interval is too small, bootloading will fail because the slave is not able to receive data before the byte-to-byte timeout expired. Refer to the SCB_CyBtldrCommRead details section for more information about byte-to-byte timeout interval and options to change it.
- Bit order: Must match Host (boot device)
- RX buffer size: Must match or be greater that maximum size of packet received from Host. The recommended RX buffer size value to select for bootloading use Bootloader Host Tool (shipped with PSoC Creator) is **64**.
- TX buffer size: Must match or be greater that maximum size of packet transmitted to the Host. The recommended TX buffer size value to select for bootloading use Bootloader Host Tool (shipped with PSoC Creator) is **64**.

For more information about the Bootloader, refer to the Bootloader component datasheet.

The following API functions are provided for Bootloader use.

SCB_CyBtldrCommStart()	Starts the SPI component and enables its interrupt.
SCB_CyBtldrCommStop()	Disables the SPI component and its interrupt.
SCB_CyBtldrCommReset()	Resets SPI receive and transmit buffers.
SCB_CyBtldrCommRead()	Allows the caller to read data from the bootloader host (the host writes the data).
SCB_CyBtldrCommWrite()	Allows the caller to write data to the bootloader host (the host reads the data).

void SCB_CyBtldrCommStart(void)

Description: Starts the SPI component and enables its interrupt (if TX or RX buffer size is greater than FIFO depth).

Every incoming SPI transfer is treated as a command for the bootloader.

Parameters: None

Return Value: None

Side Effects: None

void SCB_CyBtldrCommStop(void)

Description: Disables the SPI component and its interrupt.

Parameters: None

Return Value: None

Side Effects: None

void SCB_CyBtldrCommReset(void)

Description: Resets SPI receive and transmit buffers.

Parameters: None

Return Value: None

Side Effects: None

cystatus SCB_CyBtldrCommRead(uint8 pData[], uint16 size, uint16 * count, uint8 timeOut)

Description:	Allows the caller to read data from the bootloader host (the host writes the data). The function handles polling to allow a block of data to be completely received from the host device.
Parameters:	uint8 pData[]: Pointer to the block of data to be read from bootloader host. uint16 size: Number of bytes to be read from bootloader host. uint16 *count: Pointer to variable to write the number of bytes actually read by bootloader host. uint8 timeOut: Number of units in 10 ms to wait before returning because of a timeout.
Return Value:	cystatus: Returns CYRET_SUCCESS if no problem was encountered or returns the value that best describes the problem. For more information, refer to the “Return Codes” section of the *System Reference Guide*.
Side Effects:	None

cystatus SCB_CyBtldrCommWrite(const uint8 pData[], uint16 size, uint16 * count, uint8 timeOut)

Description:	Allows the caller to write data to the bootloader host (the host reads the data). The function handles polling to allow a block of data to be completely sent to the host device.
Parameters:	const uint8 pData[]: Pointer to the block of data to send to the bootloader host. uint16 size: Number of bytes to send to bootloader host. uint16 *count: Pointer to variable to write the number of bytes actually written to bootloader host. uint8 timeOut: Number of units in 10 ms to wait before returning because of a timeout.
Return Value:	cystatus: Returns CYRET_SUCCESS if no problem was encountered or returns the value that best describes the problem. For more information refer to the “Return Codes” section of the *System Reference Guide*.
Side Effects:	None

SCB_CyBtldrCommRead details

The SPI interface does not provide start and stop conditions to define the start and end of a transfer like I^{2}C. Therefore, the following approach is used to define when command packet from the host is received:

1) To determine when the start of a packet has occurred, the RX buffer is checked at a one millisecond interval until the buffer size is non-zero or timeout is expired. As soon as at least one data element has been received the communication component knows a packet transfer has started and immediately begins looking for the end of the packet.
2) The transfer is completed if no new data elements are received within the byte-to-byte timeout interval. This time interval is defined as the time consumed to transfer two data elements with selected data rate. It is calculated by the component based on current data

rate selection. If needed, the byte-to-byte interval can be changed by using global defines. Open project Build Settings -> Compiler -> Command line and provide global define of the interval in microseconds. For example, to change the interval to 40 microseconds:

```
-D DUT_UART_BYTE_TO_BYTE=40
```

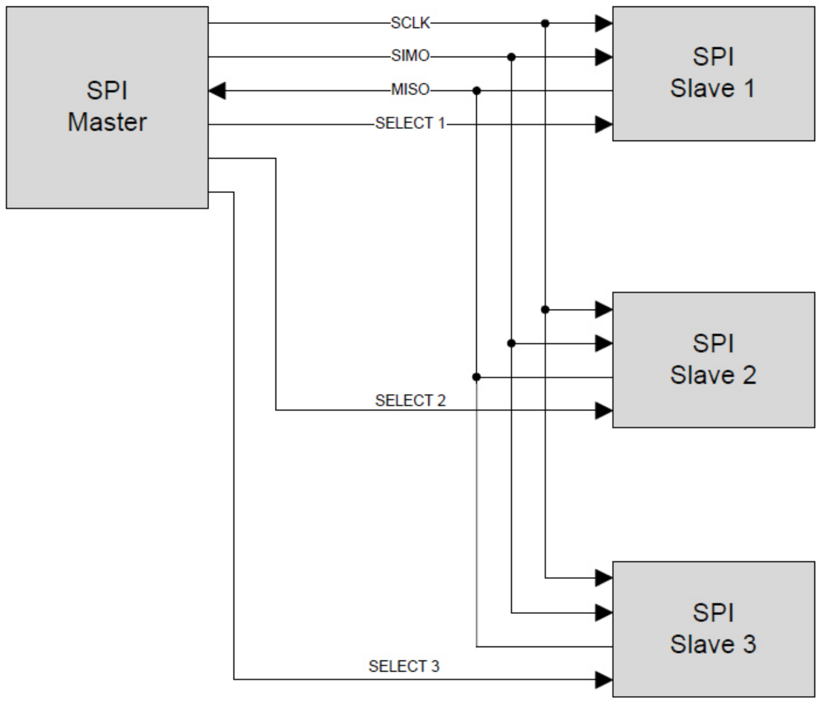
SPI Functional Description

The Serial Peripheral Interface (SPI) protocol is a synchronous serial interface, with “single-master-multi-slave” topology. Devices operate in either master or slave mode. The master initiates transfers of data frames. Multiple slaves are supported with individual slave select lines.

The SPI interface consists of four signals:

- **SCLK** – Serial clock (output from master, input to the slave).
- **MOSI** – Master output, slave input (output from the master, input to the slave).
- **MISO** – Master input, slave output (input to the master, output from the slave).
- **SELECT** – Slave select (typically an active low signal, output from the master, input to the slave).

Figure 16. SPI Bus Connections Example



Motorola sub mode operation

This is the original SPI protocol defined by Motorola. It is a full duplex protocol: transmission and reception occur at the same time.

The Motorola SPI protocol has four different modes that determine how data is driven and captured on the MOSI and MISO lines. These modes are determined by clock polarity (CPOL) and clock phase (CPHA).

- **CPHA = 0, CPOL= 0 –** Data is driven on a falling edge of SCLK. Data is captured on a rising edge of SCLK. The idle state of SCLK line is low.
- **CPHA = 0, CPOL= 1** – Data is driven on a rising edge of SCLK. Data is captured on a falling edge of SCLK. The idle state of SCLK line is high.
- **CPHA = 1, CPOL= 0** – Data is driven on a rising edge of SCLK. Data is captured on a falling edge of SCLK. The idle state of SCLK line is low.
- **CPHA = 1, CPOL= 1** – Data is driven on a falling edge of SCLK. Data is captured on a rising edge of SCLK. The idle state of SCLK line is high.

Figure 17 illustrates driving and capturing of MOSI/MISO data as a function of CPOL and CPHA.

Figure 17. SPI Motorola frame format

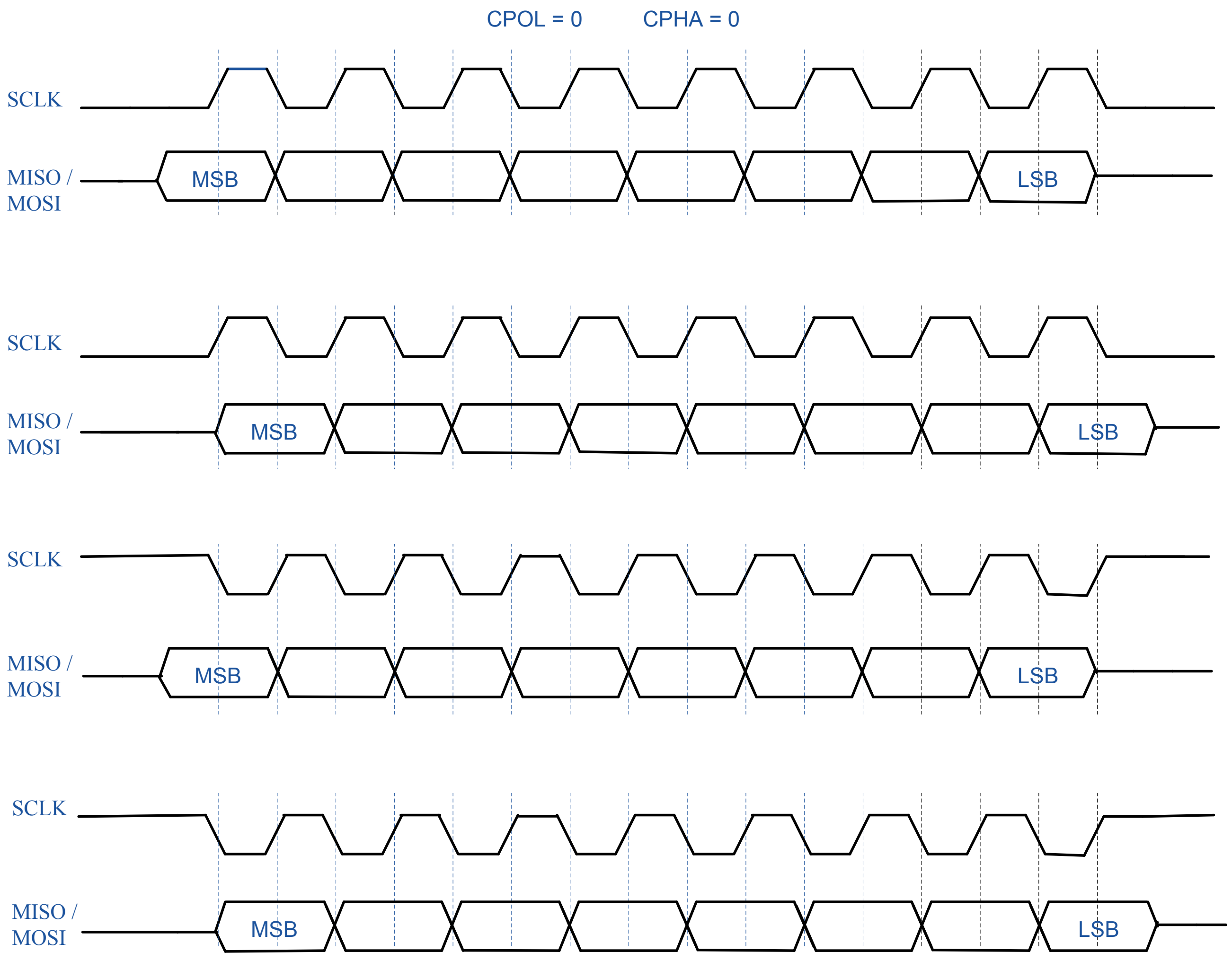


Figure 18 illustrates a single 8-bit data transfer and two successive 8-bit data transfers in mode 0 (CPOL is '0', CPHA is '0').

Figure 18. SPI Motorola Data Transfer Example

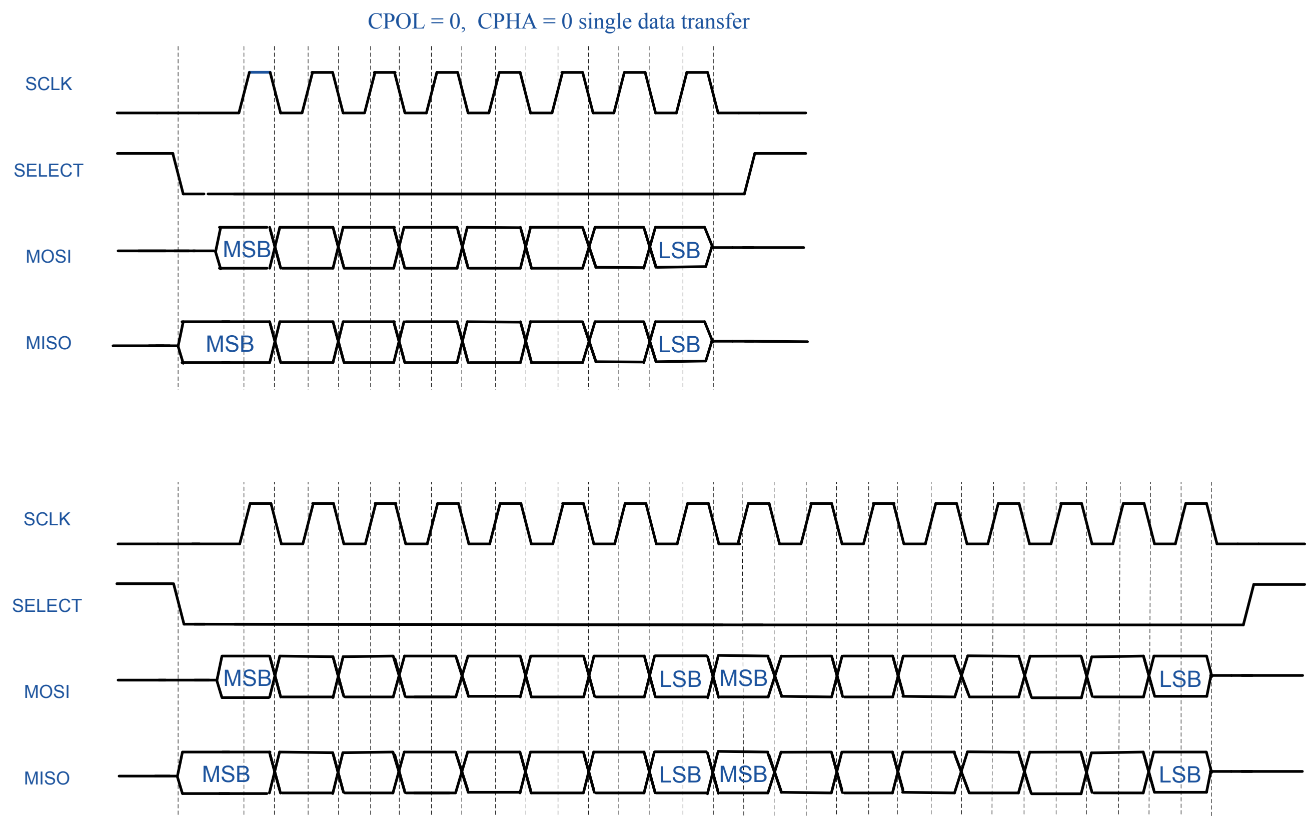


Figure 19 illustrates a single 8-bit data transfer and two successive 8-bit data transfers. The SS pulse precedes the first data bit.

Note The SELECT pulse of the second data transfer coincides with the last data bit of the first data transfer.

Figure 19. TI (Precede) Data Transfer Example

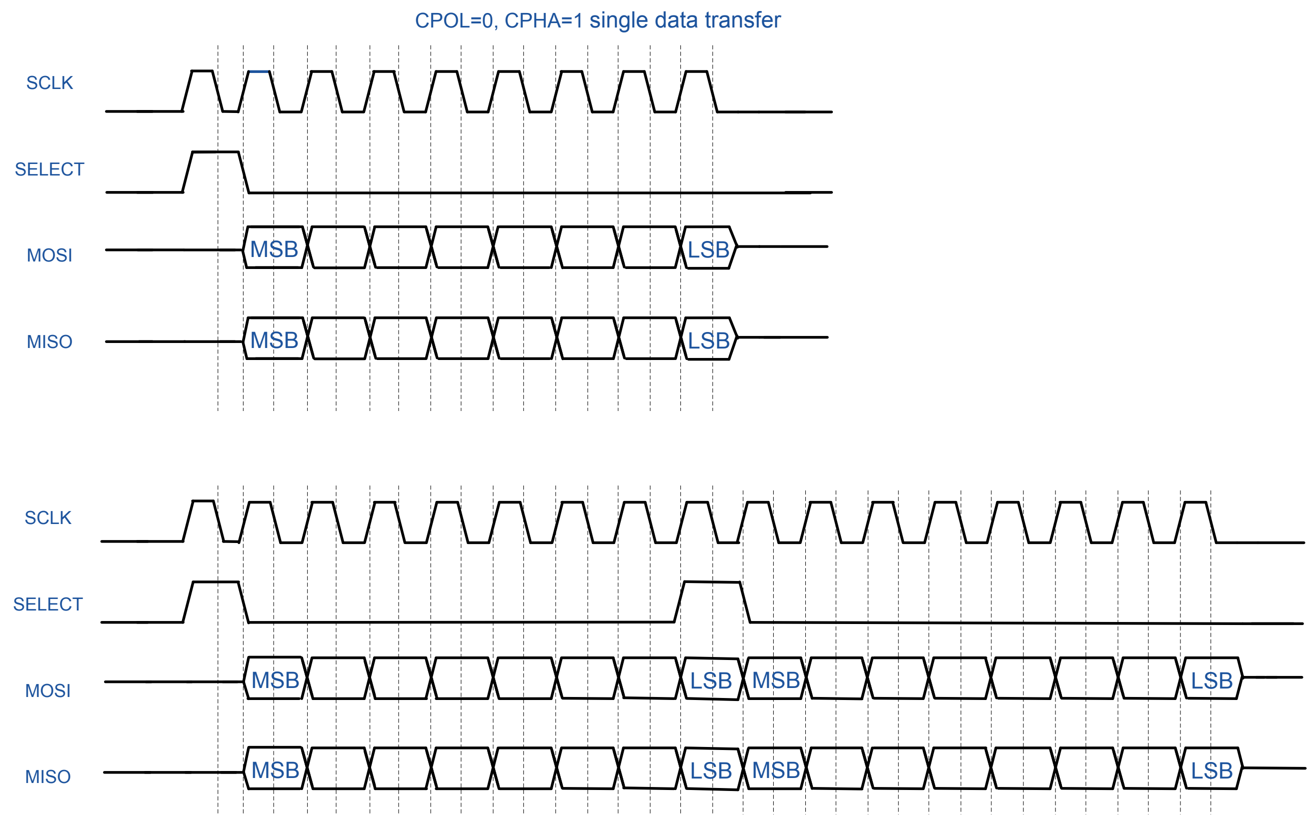


Figure 20 illustrates a single 8-bit data transfer and two successive 8-bit data transfers. The SS pulse coincides with the first data bit.

Figure 20. TI (Coincide) Data Transfer Example

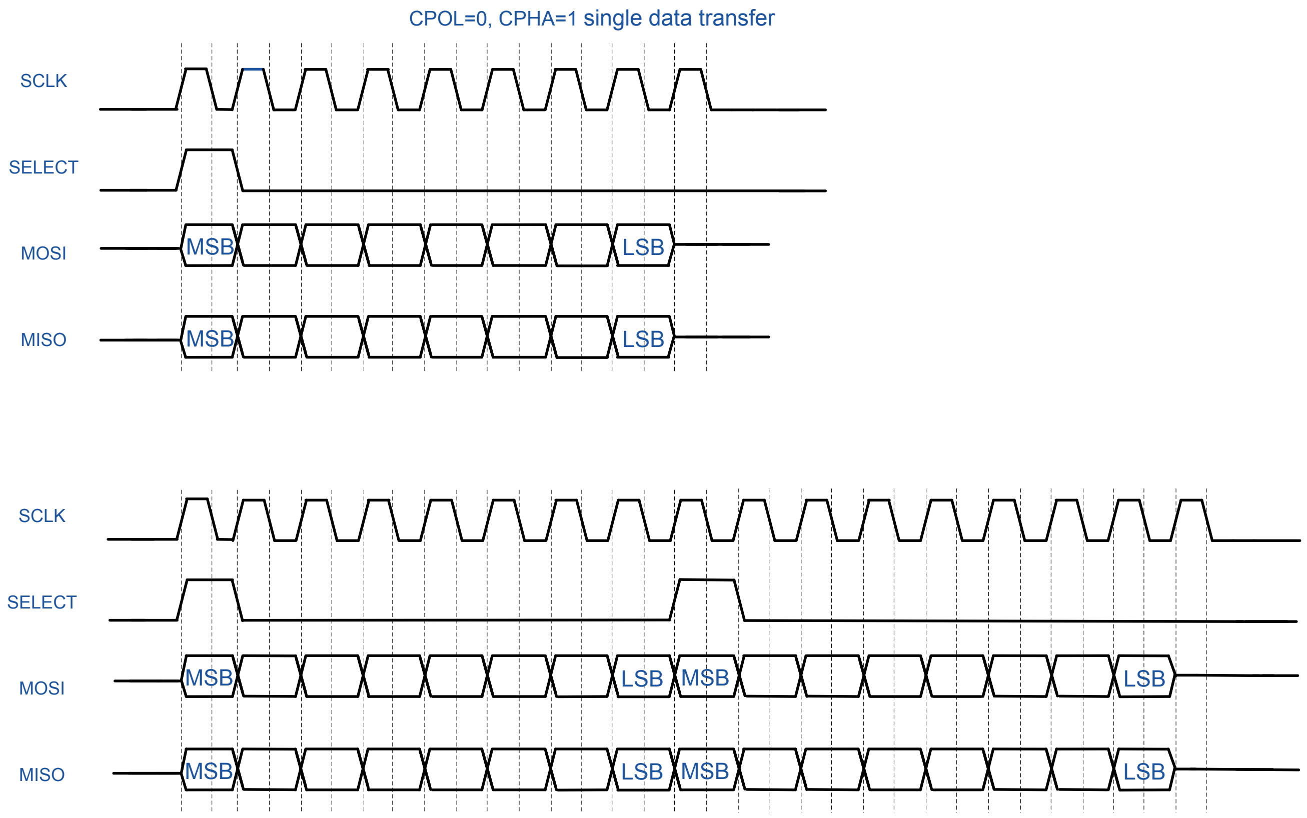
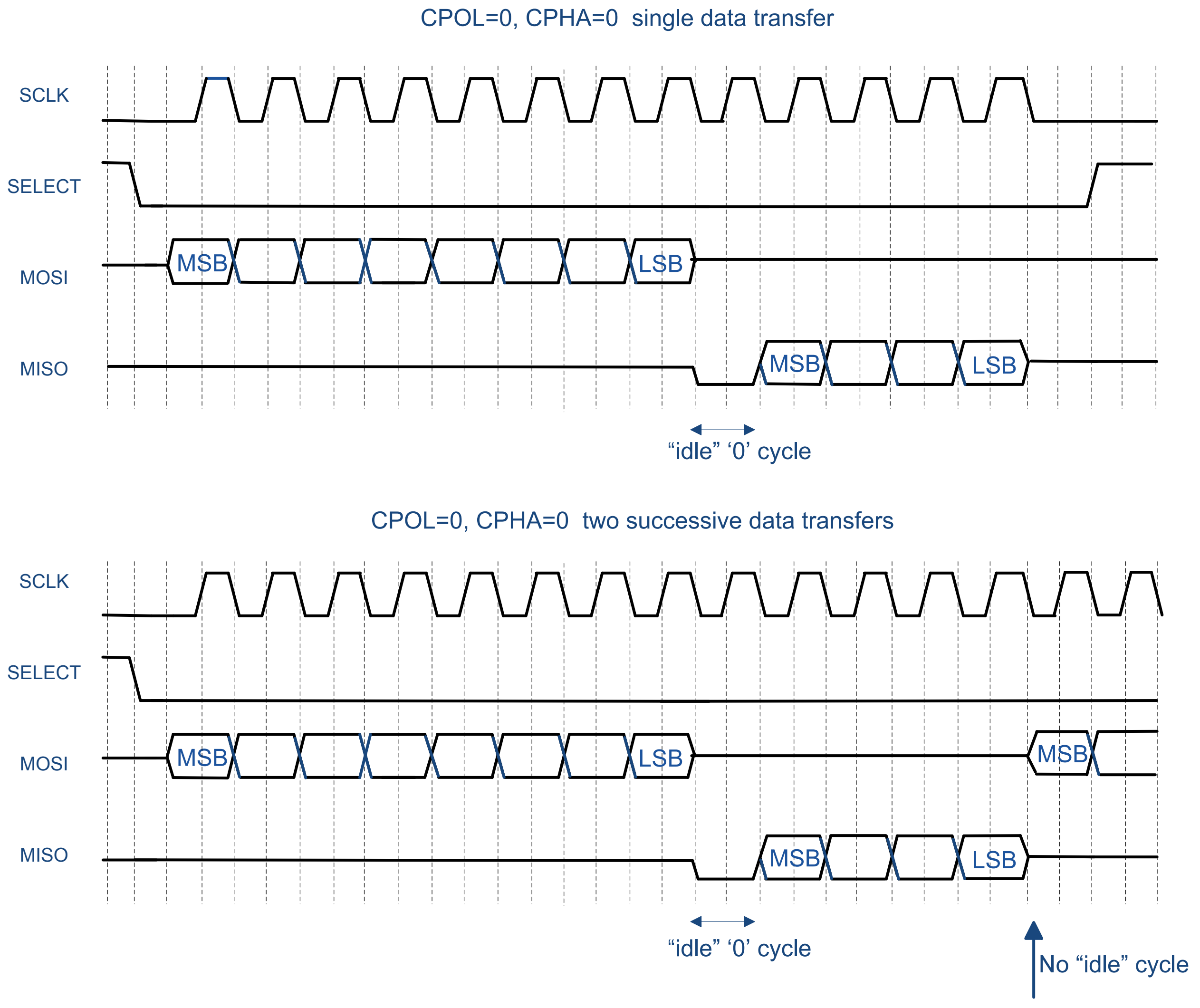


Figure 21 illustrates a single data transfer and two successive data transfers. In both cases the transmission data transfer size is 8 bits and the reception transfer size is 4 bits.

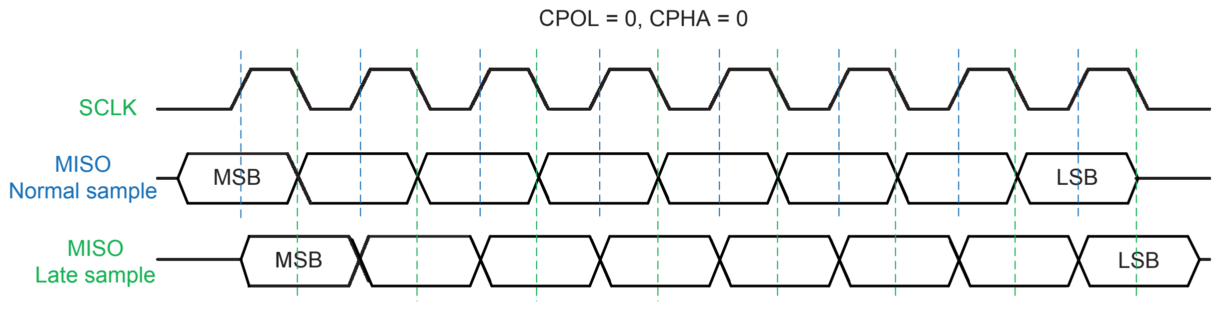
Figure 21. National Semiconductor's Microwire Data Transfer Example



MISO late sampling

The MISO is captured by Master by half of SCLK period later (on the alternate serial clock edge). Late sampling addresses the round-trip delay associated with transmitting SCLK from the master to the slave and transmitting MISO from the slave to the master.

Figure 22. Late MISO sampling example



Slave select lines

The slave select lines are used by the master to notify the slave device that it will communicate with it. The master has control of four slave select lines, and one of them has to be chosen as active before starting communication. To start communication, the data is written into the TX FIFO. The master hardware then asserts the active slave select line and sends data to the MOSI.

There are cases when firmware control of the slave select line is desired. In this case, the slave select lines that are controlled by the master hardware need to be deactivated. There are two options to do this:

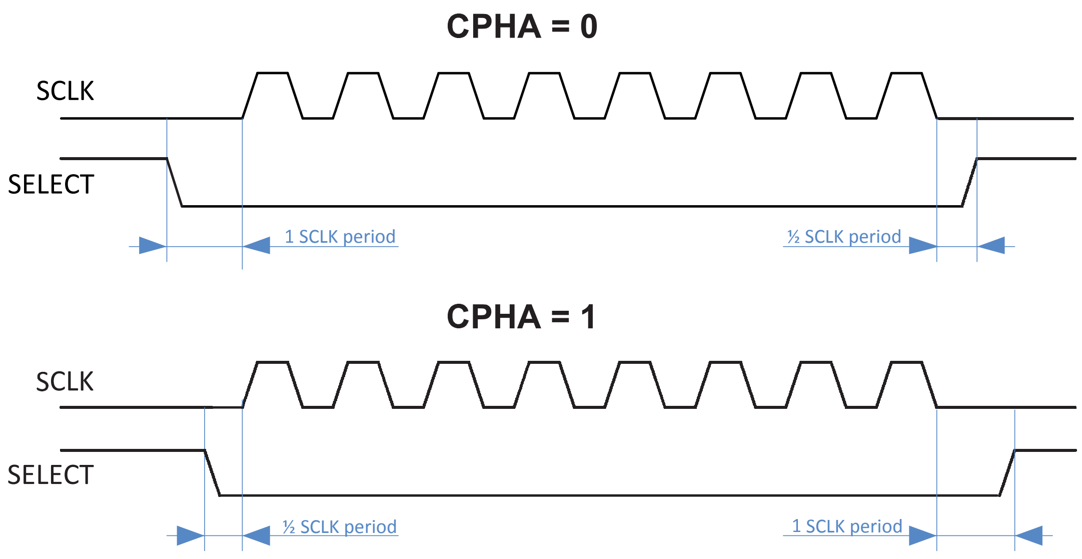
- Set the active slave select line to one that is not routed out to the pin. This option is recommended when less than four slave select lines are used by the master. Example: SPI master consumes 3 slave select lines SS0, SS1 and SS2. Call SCB_SpiSetActiveSlaveSelect(SCB_SPIM_ACTIVE_SS3) to deactivate hardware controlled slave select lines SS0-SS2.

- Change the source of the control of the active slave select line in HSIOM (High Speed I/O Matrix) from the SCB SPI interface to GPIO (CPU firmware control). Refer to the High-Speed I/O Matrix description in the *Technical Reference Manual (TRM)* for more information. This option is recommended when the master uses four slave select lines or when multiplexing between hardware controlled and firmware controlled slave select lines is required.

SELECT and SCLK Timing Correlation

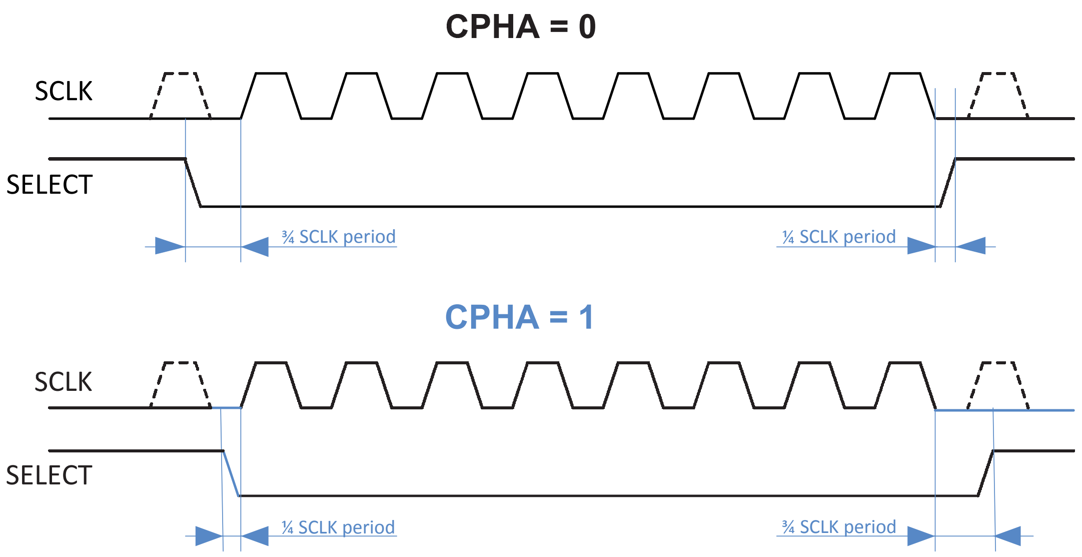
The master activates SELECT before starting the transfer and makes it inactive when the transfer is completed. A minimum time is guaranteed before SELECT activation and the first SCLK edge and SELECT deactivation and last SCLK edge. This time depends on the master sampling edge, which is defined by CPHA settings. Thus, two combinations are available.

Figure 23. SELECT and SCLK Timing Correlation (PSoC 4100/PSoC 4200)



Note PSoC 4100/PSoC 4200 devices support only SCLK gated and active low SELECT polarity.

Figure 24. SELECT and SCLK Timing Correlation (PSoC 4100 BLE/PSoC 4200 BLE/ PSoC 4100M/PSoC 4200M/PSoC 4200L)

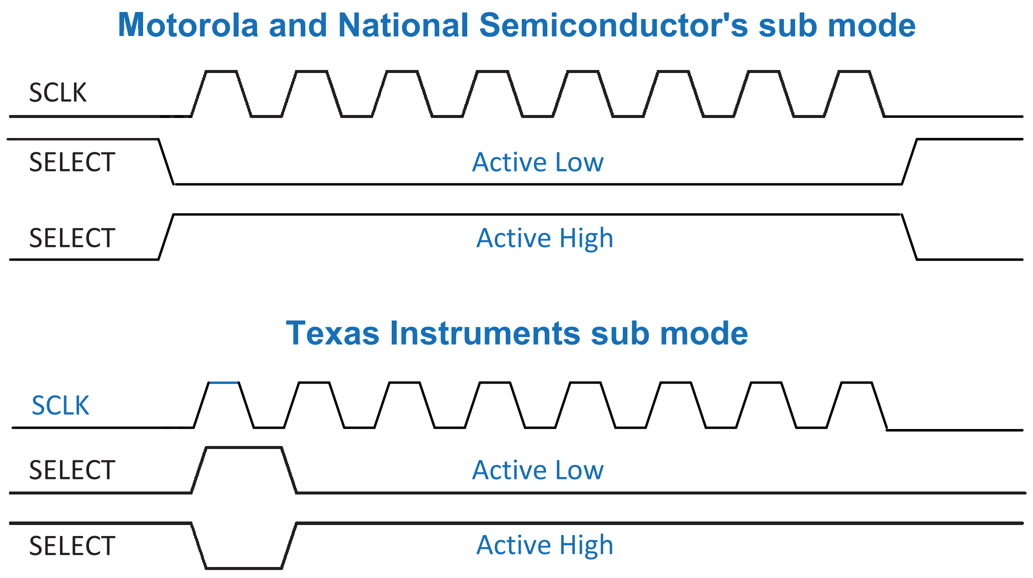


Note PSoC 4100 BLE/PSoC 4200 BLE/PSoC 4100M/PSoC 4200M/PSoC 4200L devices support SCLK gated and free running, as well as active low and high SELECT polarity. For all configurations, the same correlation is preserved.

SELECT polarity

The SELECT line polarity for PSoC 4100/PSoC 4200 devices is active low. PSoC 4100 BLE/PSoC 4200 BLE/PSoC 4100M/PSoC 4200M/PSoC 4200L devices provide the capability to select the active polarity of the line as active low or active high.

Figure 25. SELECT line polarity



Continuous versus Separated Transfer Separation

During separated data transfer, the SELECT line always changes from active to inactive state between the individual data frames until completion of the transfer.

During continuous data transfer, the individual data frame is not necessarily separated by the SELECT line inactivation. At the start of data transfer, the SELECT line is activated and keeps its state active until the end of transfer. The end of transfer is defined as all data from the TX FIFO and shifter register has been sent out. The alternative approach is to use the SPI Done interrupt source (refer to the Interrupt sources section to understand the limitations of this approach). Figure 18 on page 115 illustrates two continuous 8-bit data transfers in SCLK mode: CPHA=0, CPOL= 0.

FIFO depth

The hardware provides two FIFOs. One is used for the receive direction, RX FIFO, and the other for transmit direction, TX FIFO. The FIFO depth is 8 data elements. The width of each data element is 16 bits. The data frame width is configurable from 4-16 bits. One element from the FIFO is consumed regardless of the data frame width.

PSoC 4100 BLE/PSoC 4200 BLE/PSoC 4100M/PSoC 4200M/PSoC 4200L devices provide the ability to double the FIFO depth to be 16 data elements when the data frame width is 4-8 bits.

Software Buffer

Selecting RX or TX Buffer Size values greater than the FIFO depth enables usage of the RX or TX FIFO and a circular software buffer. An array of the requested size is allocated internally by the component for the TX software buffer. The allocated array for RX software buffer has one extra element that remains empty while in operation. Keeping this element empty simplifies circular buffer operation. The interrupt option is automatically set to Internal, and the RX or TX interrupt source is reserved to provide software buffer operation.

The internal interrupt is connected to the interrupt output. This interrupt runs a predefined interrupt service routine. Its main purpose is to provide interaction between software buffers and hardware RX or TX FIFO. The software buffer overflow can happen only for the RX direction. The data elements read from the RX FIFO that do not fit into the software buffer are discarded. This event is reported via global variable SCB_rxBufferOverflow. For the TX direction, the provided APIs do not allow the software buffer overflow.

Interrupts

When **RX buffer size** or **TX buffer size** is greater than the FIFO depth, the **RX FIFO not empty** or **TX FIFO not full** interrupt sources are reserved by the component for the internal software buffers operations. **Do not clear or disable them** because it causes incorrect software buffer operation. However, it is the user’s responsibility to clear interrupt events from other enabled interrupt sources because they are not cleared automatically. Create a custom function that clears these interrupt sources and register it using SCB_SetCustomInterruptHandler(). Each time an internal interrupt handler executes, the custom function is called before handling software buffer operation.

In case **RX buffer size** or **TX buffer size** is equal to the FIFO depth instead of software buffer only the hardware TX or RX FIFO is used. In the **Internal** interrupt mode the interrupts are not cleared automatically. It is user responsibility to do this. The **External** or **None** interrupt selection is preferred in this case.

Low power modes

The component in SPI mode is able to be a wakeup source from Sleep and Deep Sleep low power modes.

Sleep mode is identical to Active from a peripheral point of view. No configuration changes are required in the component or code before entering/exiting this mode. Any communication intended for the slave causes an interrupt to occur and leads to wakeup. Any master activity that causes an interrupt to occur leads to wakeup.

The master mode is not able to be a wakeup source from Deep Sleep. This capability is only available in slave mode. Deep Sleep mode requires that the slave be properly configured to be a wakeup source. The Enable wakeup from Deep Sleep Mode option must be checked in the SPI configuration dialog. The SCB_Sleep() and SCB_Wakeup() functions must be called before/after entering/exiting Deep Sleep.

The content of TX and RX FIFOs is cleared when device enters the Deep Sleep mode. Therefore received data should be stored in the SRAM buffer and transmit data should be transferred before enter Deep Sleep mode to avoid data lost. This is applies for **Master** and **Slave** modes.

In **Master** mode, the ongoing transfer is stopped asynchronously when the SCB_Sleep() API is called because this function disables the component. The following code sample is suggested to ensure that transfer is completed before entering Deep Sleep mode. It occurs when all data elements have been transferred from the software buffer (if utilized), TX FIFO, and shifter register. Also, the slave select line must be deactivated. This method works reliably for any choice of slave select Transfer separation configuration.

```
/* Wait until SPI Master completes transfer data */
while (0u != (SCB_SpiUartGetTxBufferSize() + SCB_GET_TX_FIFO_SR_VALID))
{
}

/* Wait until SPI Master deactivates slave select to ensure that the last
* data element has been completely tranferred.
*/
while (0u != SCB_SpiIsBusBusy())
{
}

/* SPI Master is ready to enter Deep Sleep mode. */
SCB_Sleep();
CySysPmDeepSleep();
```

In **Slave** mode, the data transmission is stopped asynchronously while the device enters Deep Sleep mode. The moment that the stop occurs depends on the Enable wakeup from Deep Sleep Mode option. When the option is disabled, the SCB_Sleep() API disables the component and the slave stops driving the MISO line at that moment. Otherwise, when the Enable wakeup from Deep Sleep Mode is enabled, the slave stops driving the MISO line at the moment when the device enters Deep Sleep mode. The previous code sample can be used for the slave as well to ensure that the transfer is completed before entering Deep Sleep mode.

The slave wakes up the device from Deep Sleep on detecting slave select activation. Waking up takes time and the ongoing SPI transfer is negatively acknowledged – "0xFF" bytes are sent out on the MISO line. The master must poll the component again after the device wake-up time is passed and data is loaded in the RX buffer.

Note The SCB_SpiUartGetTxBufferSize() or SCB_SpiUartGetRxBufferSize() function could return different values for software and hardware buffers after Deep Sleep because SRAM content is stored. However, TX and RX FIFO content is not stored while in Deep Sleep.

Slave data rate calculations

The SPI GUI calculates the actual data rate for master or slave devices. This value is based on the parameters of the component and does not take to account such factors as: parameters of external master or slave device as well as PCB delays. The master and slave parameters for PSoC4 can be found in the DC and AC Electrical Characteristics of this document or the Device datasheet.

The main factor limiting the maximum data rate between master and slave is the round trip path delay. This delay includes the PCB delay from the falling edge of SCLK at the pin of the master to the SCLK pin of the slave, the internal slave delay from the falling edge of SCLK to MISO transition, the PCB delay from the slave MISO pin to the master MISO pin, and the master setup time. The following equation takes to account delays listed above:

$t_{ROUND\_TRIP\_DELAY} = t_{SCLK\_PD\_PCB} + t_{DSO\_SLAVE} + t_{MISO\_PD\_PCB} + t_{DSI\_MASTER}$

- $t_{SCLK\_PD\_PCB}$ is the PCB path delay of SCLK from the pin of the master device to the pin of the slave device.
- t_{DSO_SLAVE} is the time it takes the slave to change MISO after SCLK clock driving edge is captured. This parameter commonly listed in the slave device datasheet.
- $t_{MISO\_PD\_PCB}$ is the PCB path delay of MISO from the pin of the slave device to the pin of the master device.
- t_{DSI_MASTER} is the setup time of MISO signal to be sampled correctly by the master (the MISO must be valid before SCLK clock capturing edge). This parameter commonly listed in the master device datasheet.

When $t_{ROUND\_TRIP\_DELAY}$ was calculated, the maximum communication data rate between master and slave can be defined as following:

f_{SCLK} (max) = 1 / (2* $t_{ROUND\_TRIP\_DELAY}$)

The assumption is made that master samples the MISO signal a half SCLK period after the driving edge.

When master is capable of sampling the MISO signal a full of SCLK period after the driving edge (late MISO sampling) the communication data rate is doubled and calculated as following:

f_{SCLK} (max) = 1 / $t_{ROUND\_TRIP\_DELAY}$

Refer to the section MISO late sampling for more information about MISO sampling by the master device.

As an example the f_{SCLK} (max) is calculated for SCB SPI Master and Slave implemented on PSoC 4100/PSoC 4200 devices. The design clock settings are following: IMO = HFCLK = SYSCLK = 48 MHz. The clock source frequency connected to the SCB SPI Slave and Master components is equal to 48MHz as well.

$t_{DSO\_SLAVE} = T_{DSO} = 42 + 3{*}t_{SCB} = 42 + 3 * (1 / 48\text{ MHz}) = 105\text{ ns}$

$t_{DSI\_MASTER} = T_{DSI} = 20$ ns (Full clock, late MISO Sampling used)

For simplicity of the calculations assume that $t_{SCLK\_PD\_PCB} = 0$ ns and $t_{MISO\_PD\_PCB} = 0$ ns.

$t_{ROUND\_TRIP\_DELAY} = t_{SCLK\_PD\_PCB} + t_{DSO\_SLAVE} + t_{MISO\_PD\_PCB} + t_{DSI\_MASTER} = 0 + 105 + 20 + 0 = 125\text{ ns}$

f_{SCLK} (max) = 1 / $t_{ROUND\_TRIP\_DELAY}$ = 1 / 125 ns = 8 MHz

The SPI master is capable to generate maximum F_{SPI} = 8 MHz and accordingly to calculation above the MISO line will be sampled properly for this data rate.

For real applications the PCB delays would need to be added, and t_{DSO_SLAVE} and t_{DSI_MASTER} adjusted to match the real master or slave device.

DMA Support

DMA is only available in PSoC 4100M/PSoC 4200M/PSoC 4200L devices.

The SPI mode provides interface to DMA controller. The signals for transmit and receive direction can be used to trigger a DMA transfer. To enable this signal, the "RX output" or "TX output" option must be enabled on the component Advanced parameter tab. The RX and TX trigger output signals are hard-wired to the DMA controller; their connection to another source will result in a build error. These signals are level sensitive and require the RX or TX FIFO level to be set. The signal behavior for the triggers is as follows:

- RX trigger output – the signal remains active until the number of data elements in the RX FIFO is greater than the value of RX FIFO level.
- TX trigger output – the signal remains active until the number of data elements in the TX FIFO is less than the value of TX FIFO level.

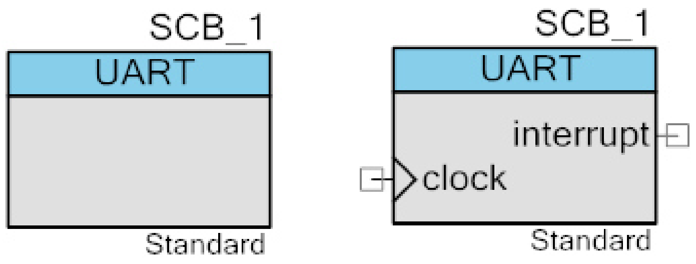
The following table specifies what DMA component configuration should be used when it is connected to the SCB (SPI mode) component.

SCB_RX_FIFO_RD_PTR	Source	Word / Byte or Halfword	rx_tr_out	Level sensitive	Receive FIFO
SCB_TX_FIFO_WR_PTR	Destination	Data bits / Byte or Halfword	tx_tr_out	Level sensitive	Transmit FIFO

Note If the number of data bits selected is less or equal to 8 bits the transfer data element width is byte, if the number of data bits is between 9 and 16 bits the width is halfword.

Note The SCB (SPI mode) clears request signal within 4 SYSCLK cycles therefore level sensitive configuration of DMA has to be "wait 4 SYSCLK".

UART



The UART provides asynchronous communications commonly referred to as RS-232. Three different UART-like serial interface protocols are supported:

- UART – this is the standard UART.
  - UART Hardware flow control
- SmartCard – similar to UART, but with the possibility to send a negative acknowledgement.
- IrDA – modification to the modulation scheme used for infrared communication.

Input/Output Connections

This section describes the various input and output connections for the SCB component. An asterisk (*) in the list of terminals indicates that the terminals may be hidden on the symbol under the conditions listed in the description of that terminals.

clock – Input*

Clock that operates this block. The presence of this terminal varies depending on the **Clock from terminal** parameter.

interrupt – Output*

This signal can only be connected to an interrupt component or left unconnected. The presence of this terminal varies depending on the **Interrupt** parameter.

rx_tr_out – Output*

This signal can only be connected to a DMA channel component. This signal is used to trigger a DMA transaction. The output of this terminal is controlled by the RX FIFO level. The presence of this terminal varies depending **RX Output** parameter.