SPI Slave Datasheet SPIS V 2.70

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/24/22/21, CY8C23x33, CY7C603xx, CY7C64215, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8CTMA140, CY8CTMA300, CY8C21x45, CY8C22x45, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx, CY8C21x12						
	1	0	0	43	0	4

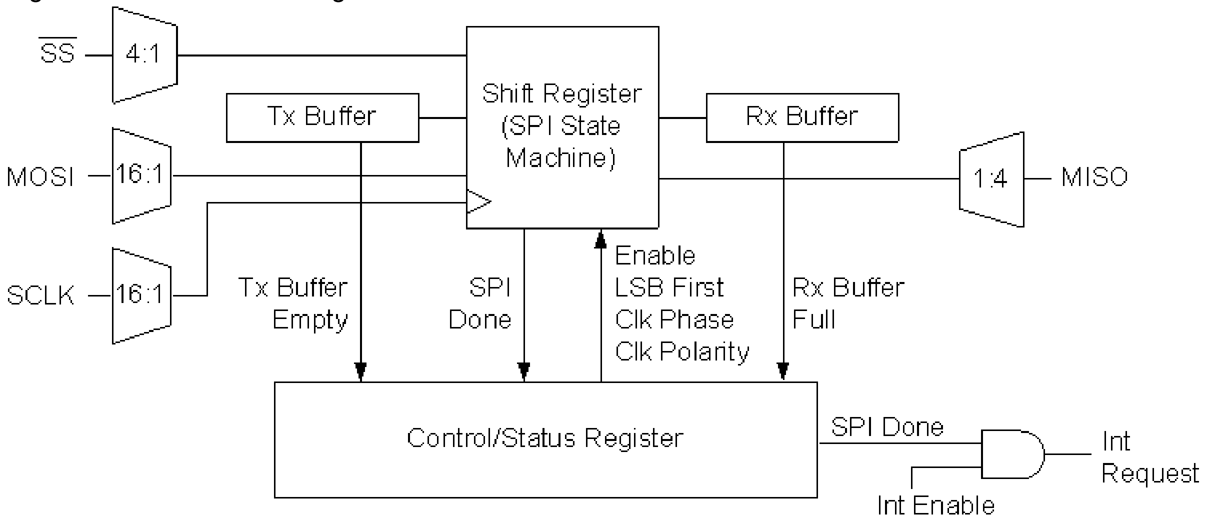
Features and Overview

- Supports serial peripheral interconnect (SPI) Slave protocol
- Supports protocol modes 0, 1, 2, and 3
- Selectable input sources for MOSI, SCLK, and ~SS
- Selectable output routing for MISO
- Programmable interrupt on SPI done condition
- SS may be firmware controlled

For more details on how SPI works in PSoC 1 devices, read the application note AN51234 - Getting Started with SPI in PSoC® 1.

The SPIS User Module is a Serial Peripheral Interconnect Slave. It performs full-duplex synchronous 8-bit data transfers. SCLK phase, SCLK polarity, and LSB First can be specified to accommodate most SPI protocols. The SPIS PSoC block has selectable routing for the input and output signals, and programmable interrupt driven control. Application Programming Interface (API) firmware provides a high-level programming interface for either assembly or C application software.

Figure 1. SPIS Block Diagram



Functional Description

SPIS is a user module that implements a Serial Peripheral Interconnect Slave. It uses the Tx Buffer, Rx Buffer, Control, and Shift registers of a Digital Communications Type PSoC block.

The Control register is initialized and configured using the Device Editor and/or the SPIS User Module firmware Application Programming Interface (API) routines. Initialization includes setting LSB First and the SPI transmission/receive protocol modes. SPI modes 0, 1, 2, and 3 are supported. Both the SPI Master and SPI Slave must be set with the same mode and bit configuration in order to properly communicate. The SPI modes are defined as follows.

Table 1. SPI Modes

Mode	SCLK Edge Performing Data Latch	Clock Polarity	Notes
0	Leading	Non-inverting	Leading edge latches data. Data changes on trailing edge of clock.
1	Leading	Inverted	
2	Trailing	Non-inverting	Trailing edge latches data. Data changes on leading edge.
3	Trailing	Inverted	

Note Check to ensure that the clock polarity and phase for the selected mode match the settings used for any attached SPI devices, as mode numbers may not be the same for all devices.

The SCLK input signal is the SPI transmit/receive clock generated by the SPI Master. It defines the bit rate of the transmitted/received data.

The MOSI input signal is the Master-Out-Slave-In data signal that receives the data from the SPI Master.

The MISO output signal is the Master-In-Slave-Out data signal that transmits the data from the Shift register to the SPI Master device. Typically, the MISO signals of multiple slaves are tied together. Each slave tri-states its respective MISO signal, until its Slave Select signal is asserted. This user module does not tri-state the MISO output signal. The user can easily add this functionality to the user module if required.

The SPIS hardware receives data from the Master SPI device on the MOSI signal and simultaneously transmits data to the SPI Master device on the MISO signal. The same SCLK signal is used for both transmit and receive of the master and slave data.

The SPI protocol is a master-only initiated response protocol. The master asserts the Slave Select (~SS) input signal low, to enable the specific SPIS device for active communication.

It is the master’s responsibility to determine if the selected slave device is ready for a command or is ready to receive data.

The SPIS User Module is enabled for operation when the SPI Enable bit is set in the Control register using an API routine.

Data to be transmitted to the SPI Master is written to the Tx Buffer register. This clears the Tx Buffer Empty status bit.

On the falling edge of the Slave Select signal, the data is transferred from the Tx Buffer register to the Shift register. The first bit of the data byte to be transmitted is then asserted on the MISO output signal. Another data byte to transmit can be written to the Tx Buffer register at this time. Upon completion of the transmission of the current byte, this data will be ready to send to the SPI master.

On the assertion of each SCLK input signal, the data is simultaneously shifted out of the Shift register to the MISO output and shifted into the Shift register from the MOSI input. The specific timing of the SCLK, MOSI, and MISO signals will be based on the SPI mode configuration.

After all of the bits have been transmitted and simultaneously received, the received data is transferred from the Shift register to the Rx Buffer register and the Tx Buffer register is transferred to the Shift register. The Rx Buffer Full and the SPI Done status bits are set. If the interrupt is enabled, the SPI Done status bit will cause it to trigger. This interrupt can be used to alert the software that a byte of data has been received or that a byte was successfully transmitted.

If a pending byte of data is currently loaded in the Tx Buffer register, then this byte will be ready to transmit when the master performs the next transaction.

If the SPI Done interrupt condition is not used to retrieve the data byte from the Rx Buffer register, the Control register should be polled to monitor the Rx Buffer Full status bit. The received data must be read from the Rx Buffer register before the next data byte is fully received or the Overrun Error status bit will be set.

The SPI Done bit should be monitored to determine when to disable the SPIS User Module. This guarantees that all of the clocking signals have completed between the master and slave SPI devices.

If interrupts are used, the SPIS status register must be cleared following each interrupt for the next interrupt to be recognized. Reading the status register clears the internal signal that is recognized by the interrupt controller. If this signal remains high, subsequent SPIS interrupts will be masked. This applies to the TxComplete and RxComplete interrupts, not to the TxRegEmpty and RxRegEmpty interrupts. Either the assembly language **MOV** or **TST** opcode can be used to read and clear the register.

Timing

Typical timing for a SPIS transfer is shown in the following diagrams. See the Technical Reference Manual for more SPIS timing information.

Figure 2. Typical SPIS Timing for Modes 0 and 1

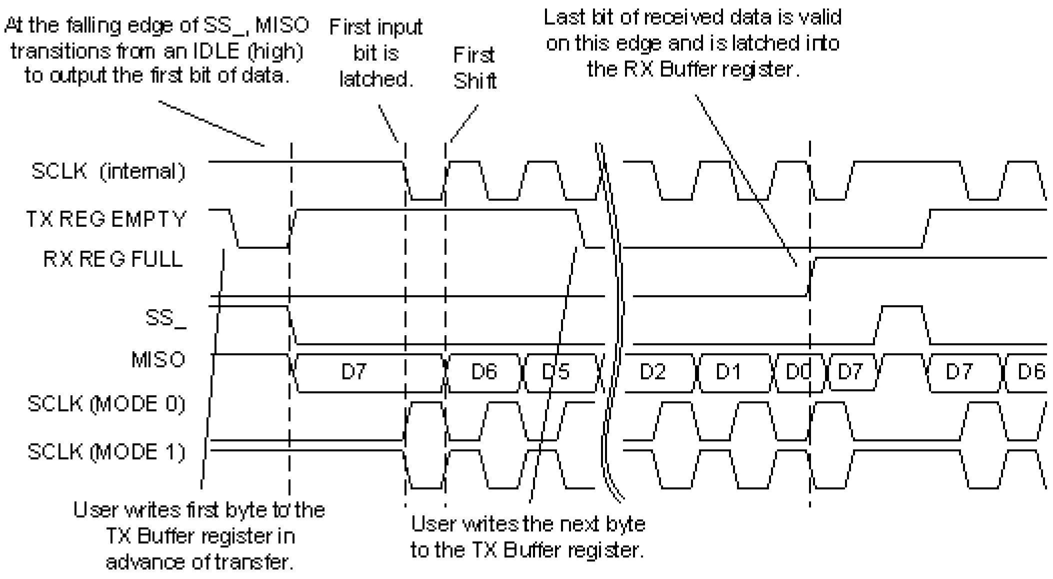
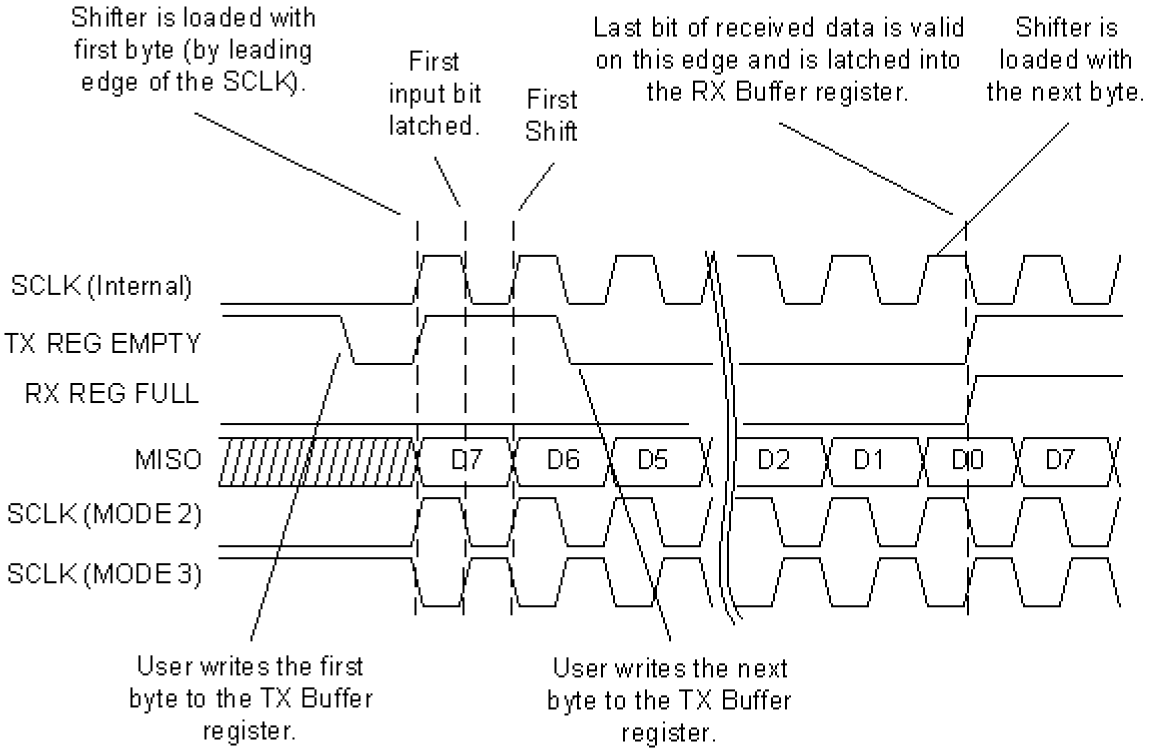

At the falling edge of SS_, MISO transitions from an IDLE (high) to output the first bit of data.
First input bit is latched.
First Shift
Last bit of received data is valid on this edge and is latched into the RX Buffer register.
SCLK (internal)
TX REG EMPTY
RX REG FULL
SS_
MISO
SCLK (MODE 0)
SCLK (MODE 1)
D7
D6
D5
D2
D1
D0
D7
D7
D6
User writes first byte to the TX Buffer register in advance of transfer.
User writes the next byte to the TX Buffer register.

Figure 3. Typical SPIS Timing for Modes 2 and 3


Shifter is loaded with first byte (by leading edge of the SCLK).
First input bit latched.
First Shift
Last bit of received data is valid on this edge and is latched into the RX Buffer register.
Shifter is loaded with the next byte.
SCLK (Internal)
TX REG EMPTY
RX REG FULL
MISO
SCLK (MODE 2)
SCLK (MODE 3)
D7
D6
D5
D2
D1
D0
D7
User writes the first byte to the TX Buffer register.
User writes the next byte to the TX Buffer register.

DC and AC Electrical Characteristics

Table 2. SPIS DC and AC Electrical Characteristics

Parameter	Conditions and Notes	Typical	Limit	Units
F_{InMax}	Maximum Input SCLK frequency	4.1	8.2	MHz
F_{SCLK}	SCLK clock frequency		5	MHz
T_{SCLK_MISO}	SCLK to MISO valid		100	ns

Placement

The SPIS maps onto a single PSoC block. The block name is SPIS in the PSoC Designer Device Editor. It may be placed in any of the Digital Communications blocks.

Parameters and Resources

SCLK

SPIS is clocked by the SPI Master generated SCLK signal. This signal can be routed from one of the available sources. High, low, the global I/O buses, the analog comparator buses, or another PSoC block can be specified as supplying the SCLK input. This clock defines the effective bit transfer rate. By default SCLK is synchronized to SysClk. If SCLK is to be left unsynchronized or synchronized to SysClk*2 then a direct write to the ClockSync bitfield of the Output Register should be used. Available clocks and clock rates will vary by PSoC device.

MOSI

The Master-Out-Slave-In input signal can be routed from one of 16 possible sources. High, low, the global I/O buses, the analog comparator buses, or another PSoC block can be specified as supplying the MOSI input.

Note: The Row Input synchronization for MOSI should be set to Async for High SPI data rates >1MHz.

Slave Select Input

The Slave Select input signal can be routed from one of four possible global inputs. In addition, the signal may also be firmware controlled. When using the SW_SlaveSelect, the Slave Select signal will assert on startup of the user module. This instance is not observed when using a hardware Slave Select Input.

MISO

The Master-In-Slave-Out output signal can be routed to one of the global output buses. The global output bus can then be connected to an external pin or to another PSoC block for further processing.

Interrupt Mode

This option determines when an interrupt will be generated for the TX block. The “TxRegEmpty” option causes an interrupt to be generated as soon as the data has been transferred from the Data register to the Shift register. Choosing the second option, “TxComplete,” delays the interrupt until the last bit is shifted out of the Shift register. This second option is useful in that it is important to know when the character has been completely sent. The first option, “TxRegEmpty,” is best used to maximize the output of the transmitter. It allows a byte to be loaded while the previous byte is being sent.

InvertMOSI

This parameter gives the user the option to invert the MOSI input.

ClockSync

This parameter defines the clock source with which SPIS signals are synchronized. The following table provides appropriate values for this parameter:

ClockSync Value	Use
Sync to SysClk	Use this setting to synchronize the SPIS signals with the SysClk clock
Sync to SysClk*2	Use this setting to synchronize the SPIS signals with the SysClk*2 clock
Unsynchronized	Use this setting to not synchronize the SPIS signals with the SysClk clock.This setting is required for SPIS to operate in sleep mode.

Interrupt Generation Control

There are two additional parameters that become available when the **Enable interrupt generation control** check box in PSoC Designer is checked. This is available under **Project > Settings > Chip Editor**. Interrupt Generation Control is important when multiple overlays are used with interrupts shared by multiple user modules across overlays:

- Interrupt API
- IntDispatchMode

InterruptAPI

The InterruptAPI parameter allows conditional generation of a user module's interrupt handler and interrupt vector table entry. Select "Enable" to generate the interrupt handler and interrupt vector table entry. Select "Disable" to bypass the generation of the interrupt handler and interrupt vector table entry. Properly selecting whether an Interrupt API is to be generated is recommended particularly with projects that have multiple overlays where a single block resource is used by the different overlays. By selecting only Interrupt API generation when it is necessary the need to generate an interrupt dispatch code might be eliminated, thereby reducing overhead.

IntDispatchMode

The IntDispatchMode parameter is used to specify how an interrupt request is handled for interrupts shared by multiple user modules existing in the same block but in different overlays. Selecting "ActiveStatus" causes firmware to test which overlay is active before servicing the shared interrupt request. This test occurs every time the shared interrupt is requested. This adds latency and also produces a nondeterministic procedure of servicing shared interrupt requests, but does not require any RAM. Selecting "OffsetPreCalc" causes firmware to calculate the source of a shared interrupt request only when an overlay is initially loaded. This calculation decreases interrupt latency and produces a deterministic procedure for servicing shared interrupt requests, but at the expense of a byte of RAM.

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the “include” files.

Note

In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X prior to the call if those values are required after the call. This “registers are volatile” policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API function may leave A and X unchanged, there is no guarantee they will do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

This section lists the list of SPIS supplied API functions:

SPIS_Start

Description:

Sets the mode configuration of the SPI interface and enables the SPIS module by setting the proper bits in the Control register. Prior to calling this function, all of the slave select signal(s) should be asserted high to deselect connected SPI Slave devices. This should be done in a user-supplied routine.

C Prototype:

```
void  SPIS_Start(BYTE bConfiguration)
```

Assembly:

```
mov   A, SPIS_SPIS_MODE_2 | SPIS_SPIS_LSB_FIRST
lcall  SPIS_Start
```

Parameters:

bConfiguration: One byte that specifies the SPI mode and LSB First configurations. Symbolic names provided in C and assembly, and their associated values, are given in the following table. Note that the symbolic names can be OR’ed together to form the configuration of the SPI interface. Also note that the instance name of the user module is prepended to the symbolic name listed below. For example, if you named the user module SPIS1 when you placed it, the symbolic name of the first mode is SPIS1_SPIS_MODE_0.

Symbolic Name	Value
SPIS_SPIS_MODE_0	0x00
SPIS_SPIS_MODE_1	0x02

Symbolic Name	Value
SPIS_SPIS_MODE_2	0x04
SPIS_SPIS_MODE_3	0x06
SPIS_SPIS_LSB_FIRST	0X80
SPIS_SPIS_MSB_FIRST	0X00

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

SPIS_Stop

Description:

Disables the SPIS module by clearing the enable bit in the Control register.

C Prototype:

```
void SPIS_Stop(void)
```

Assembly:

```
lcall  SPIS_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

SPIS_EnableInt

Description:

Enables the SPIS interrupt on the SPI Done condition. The placement location of the SPIS determines the specific interrupt vector and priority.

C Prototype:

```
void SPIS_EnableInt(void)
```

Assembly:

```
lcall  SPIS_EnableInt
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

SPIS_DisableInt

Description:

Disables the SPIS interrupt on the SPI Done condition.

C Prototype:

```
void SPIS_DisableInt(void)
```

Assembly:

```
lcall  SPIS_DisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

SPIS_SetupTxData

Description:

Writes the data byte to transmit to the SPI Master into the Tx Buffer register.

C Prototype:

```
void SPIS_SetupTxData(BYTE bTxData)
```

Assembly:

```
mov   A, bTxData
lcall  SPIS_SetupTxData
```

Parameters:

bTxData: Data to be sent to the SPI Master device and passed in the Accumulator.

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

SPIS_bReadRxData

Description:

Returns a received data byte from a slave device. The Rx Buffer Full flag should be checked prior to calling this routine, to verify that a data byte has been received.

C Prototype:

```
BYTE SPIS_bReadRxData(void)
```

Assembly:

```
lcall  SPIS_bReadRxData
mov   bRxData, A
```

Parameters:

None

Return Value:

Data byte received from the slave SPI and returned in the Accumulator.

Side Effects:

The A and X registers may be altered by this function.

SPIS_bReadStatus

Description:

Reads and returns the current SPIS Control/Status register.

C Prototype:

```
BYTE SPIS_bReadStatus(void)
```

Assembly:

```
lcall  SPIS_bReadStatus
and   A, SPIS_SPIS_SPI_COMPLETE | SPIS_SPIS_RX_BUFFER_FULL
jnz   SPI_COMPLETE_GET_RX_DATA
```

Parameters:

None

Return Value:

Returns status byte read and is returned in the Accumulator. Utilize defined masks to test for specific status conditions. Note that masks can be OR'ed together to test for multiple conditions.

SPIS Status Masks	Value
SPIS_SPIS_SPI_COMPLETE	0x20
SPIS_SPIS_RX_OVERRUN_ERROR	0x40
SPIS_SPIS_TX_BUFFER_EMPTY	0x10
SPIS_SPIS_RX_BUFFER_FULL	0x08

Side Effects:

The status bits are cleared after this function is called. The A and X registers may be altered by this function.

SPIS_DisableSS

Description:

Sets the active-low Slave Select signal (~SS) to the high state using firmware when an external ~SS signal is not required. In order to use this function, the SPI parameter named “Slave Select Input” must be set to the value of “SW_SlaveSelect,” rather than one of the row inputs. If an external signal is connected, this function may not be effective.

C Prototype:

```
void SPIS_DisableSS(void)
```

Assembly:

```
lcall SPIS_DisableSS
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

SPIS_EnableSS

Description:

Sets the active-low Slave Select signal (~SS) to the low state using firmware when an external ~SS signal is not required. In order to use this function, the SPI parameter named “Slave Select Input” must be set to the value of “SW_SlaveSelect,” rather than one of the row inputs. If an external signal is connected, this function may not be effective.

C Prototype:

```
void SPIS_EnableSS(void)
```

Assembly:

```
lcall SPIS_EnableSS
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

Sample Firmware Source Code

This example shows how to create an SPI loop back. It echoes the data received from the SPI Master. If an error condition is detected, a NAK is sent:

```
;*****************************************************************
;  Loop Back:
;
;  This code sample illustrates how to setup a SPI Slave loop back.
;  Data received is echoed back to the SPI master.
;
;  This sample is written to be performed in the non-interrupt
;  processing.  This code could easily be written to be
;  interrupt driven.
;
;  It is assumed that the SPI Slave User Module is setup properly
;  and started.
;
;*****************************************************************
include  "m8c.inc"       ; part specific constants and macros
include  "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include  "PSoCAPI.inc"   ; PSoC API definitions for all user modules

export   LoopBack
export   _main

_main:
        mov   A, SPIS_SPIS_MODE_2 | SPIS_SPIS_LSB_FIRST
        lcall SPIS_Start
        call  LoopBack
LoopBack:
        ; wait for data to be received
        lcall SPIS_bReadStatus
        and   A, SPIS_SPIS_SPI_COMPLETE
        jz    LoopBack

        ; read the data from the receiver
        lcall SPIS_bReadRxData

        ; setup to transmit the response data
        lcall SPIS_SetupTxData

        ; go wait for next byte
        jmp   LoopBack
```

The same code written in C is:

```
#include  <m8c.h>        // part specific constants and macros
#include  "PSoCAPI.h"    // PSoC API definitions for all user modules

void  LoopBack(void)
{
    BYTE  bData;
    while(1)
    {
        /* wait for data to be received */
        while( !( SPIS_bReadStatus() & SPIS_SPIS_SPI_COMPLETE ) );

        /* read the received data */
        bData = SPIS_bReadRxData();


        /* setup to transmit the response data */
        SPIS_SetupTxData(bData);
    }
}
void main(void)
{
    SPIS_Start(SPIS_SPIS_MODE_2 | SPIS_SPIS_LSB_FIRST);
    LoopBack();
}
```

Configuration Registers

The Digital Communication Type A PSoC block registers used to configure this user module are described below. Only the parameterized symbols are explained.

Table 3. Block SPIS: Register Function

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	1	1	1	0

This register defines the personality of this Digital Communications Type ‘A’ Block to be a SPIS User Module.

Table 4. Block SPIS: Register Input

Bit	7	6	5	4	3	2	1	0
Value	MOSI				SCLK			

MOSI is the Master-Out-Slave-In input signal. SCLK is the clock signal from the SPI Master. Both are set using the Device Editor during parameter selection.

Table 5. Block SPIS: Register Output

Bit	7	6	5	4	3	2	1	0
Value	0	ClockSync		~SS		MISO		

~SS is the Slave Select input signal. MISO is the Master-In-Slave-Out output signal. Both are set using the Device Editor during parameter selection.

Table 6. Block SPIS: Shift Register DR0

Bit	7	6	5	4	3	2	1	0
Value	Shift Register							

Shift Register is a SPI Shift register.

Table 7. Block SPIS: TX Data Buffer Register DR1

Bit	7	6	5	4	3	2	1	0
Value	TX Buffer Register							

TX Buffer Register: data written to this buffer will be transferred to the Shift register when the PSoC block is enabled.

Table 8. Block SPIS: RX Data Buffer Register DR2

Bit	7	6	5	4	3	2	1	0
Value	RX Buffer Register							

RX Buffer Register: data received in the Shift register is transferred to this register after completion of the SPI transmit cycle.

Table 9. Block SPIS: Control Register CR0

Bit	7	6	5	4	3	2	1	0
Value	LSB First	RX Overrun Error	SPI Done	TX Buffer Empty	RX Buffer Full	Clock Phase	Clock Polarity	SPIS Enable

LSB First specifies that the LSB bit should be transmitted first.

RX Overrun Error is a flag that indicates that the previously received data byte was not read before the next byte was received.

SPI Done is a flag that indicates that the complete SPI transmit/receive cycle has been completed.

TX Buffer Empty is a flag that indicates that the TX Buffer is empty.

RX Buffer Full is a flag that indicates that a byte of data has been received from the Shift register.

Clock Phase indicates the phase of the SCLK signal. It is one of the parameters that defines the SPI mode.

Clock Polarity indicates the polarity of the SCLK signal. It is one of the parameters that defines the SPI mode.

SPIS Enable enables the SPIS PSoC block when set.

Version History

Version	Originator	Description
2.5	DHA	Updated Slave Select parameter.
2.60	DHA	Added support for CY8C21x12 devices.
2.60.b	DHA	Updated MISO parameter description.
2.70	DHA	Added user module parameter "ClockSync".

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.