SPI Master Datasheet SPIM V 1.30

Resources	API Memory (Bytes)		Pins
	Flash	RAM	
CY7C639/638/633/602/601xx, CYRF69xx3			
MOSI/MISO/SCLK	38	0	3 - 4
SDIO/SCK	38	0	2 - 3

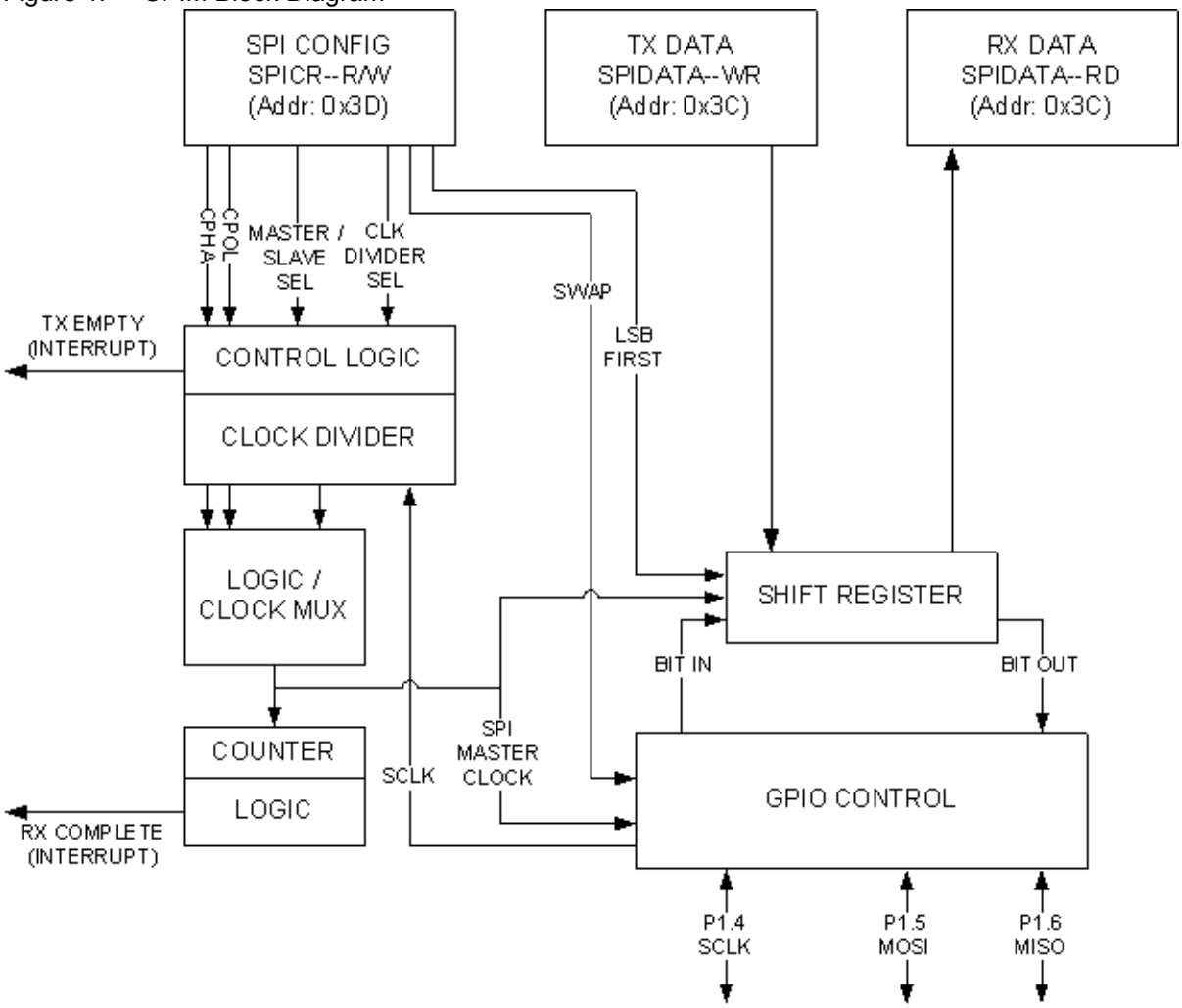
Features and Overview

- Supports Serial Peripheral Interconnect (SPI) Master protocol
- Supports SPI clocking modes 0, 1, 2, and 3
- Programmable interrupt on SPI-done condition
- SPI Slave devices can be independently selected

The SPIM User Module is a serial peripheral interconnect master. It performs full-duplex synchronous 8-bit data transfers. SCLK phase, SCLK polarity, and LSB First can be specified to accommodate most SPI clocking modes. Controlled by user-supplied software, you can configure the slave select signal to control one or more SPI Slave devices.

The SPI User Module also supports two-wire serial devices such as optical mouse sensors that employ an SDIO/SCK interface.

Figure 1. SPIM Block Diagram



Parameters and Resources

SPIPins

The SPIPins User Module Parameter selects the pins to be used by the SPI block. The parameter allows two-wire (SDIO/SCK) or three-wire (MOSI/MISO/SCLK) operation. If the selected device supports both 5-V and 3.3-V operation, the parameter also provides voltage selection.

The SPIPins parameter programs the P14CR, P15CR, and P16CR configuration registers.

SPIPins Choices	Voltage Options	
	CY7C639xx CY7C638xx CY7C633xx	CY7C601xx CY7C602xx
MOSI(P1.5)/MISO(P1.6)/SCLK(P1.4)	5 V or 3.3 V	N/A
SDIO(P1.5)/SCK(P1.4)	5 V or 3.3 V	N/A
The CY7C601xx and CY7C602xx devices do not offer dual voltage support for the SPI pins.		

BitOrder

The BitOrder user module parameter selects the bit order for the SPI data stream. Selections are LSBFirst or MSBFirst.

The BitOrder user module parameter programs the LSB First field of the SPI Configuration Register (SPICR).

CPOL

The CPOL user module parameter selects the idle state for the SPI Clock.

The CPOL user module parameter programs the CPOL field of the SPI Configuration Register (SPICR)

CPHA

The CPHA user module parameter selects SPI Clock Phase on which the SPI data is sampled.

The CPHA user module parameter programs the CPHA field of the SPI Configuration Register (SPICR).

ClockDivider

The ClockDivider user module parameter selects SPI Clock ClockDivider. The ClockDivider is applied to the CPUCLK to generate the SPI Clock.

ClockDivider	SCLK Select	SCLK Frequency when CPUCLK =	
		12 MHz	24 MHz
6	00	2 MHz	4 MHz
12	01	1 MHz	2 MHz
48	10	250 kHz	500 kHz
96	11	125 kHz	250 kHz

The ClockDivider user module parameter programs the SCLK Select field of the SPI Configuration Register (SPICR).

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.

SPIM_Start

Description:

Starts the SPIM User Module. This function is provided for API consistency.

C Prototype:

```
void  SPIM_Start(void)
```

Assembly:

```
lcall  SPIM_Start
```

Parameters:

None

Return Value:

None

Side Effects:

You can alter the A and X registers by this function.

SPIM_Stop

Description:

Disables the SPIM module. This function is provided for API consistency.

C Prototype:

```
void  SPIM_Stop(void)
```

Assembly:

```
lcall  SPIM_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

You can alter the A and X registers by this function.

SPIM_EnableInt

Description:

Enables the SPIM TX Buffer Full and RX Complete interrupts.

Note You can add the custom ISR code to the ISRs found in the SPIMINT.asm file in the project lib directory.

C Prototype:

```
void  SPIM_EnableInt(void)
```

Assembly:

```
lcall  SPIM_EnableInt
```

Parameters:

None

Return Value:

None

Side Effects:

You can alter the A and X registers by this function.

SPIM_DisableInt

Description:

Disables the SPIM TX Buffer Full and RX Complete interrupts.

C Prototype:

```
void  SPIM_DisableInt(void)
```

Assembly:

```
lcall  SPIM_DisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

You can alter the A and X registers by this function.

SPIM_SetMOSI

Description:

Sets the output pin for Master Out, Slave In.

C Prototype:

```
void  SPIM_SetMOSI(BYTE bPin)
```

Assembly:

```
mov   A, bPin
lcall  SPIM_SetMOSI
```

Parameters:

BYTE bPin: 0x00--SPIM_MOSI_P15, 0x01--SPIM_MOSI_P16. It is passed in Accumulator.

Return Value:

None

Side Effects:

You can alter the A and X registers by this function.

SPIM_SetMISO

Description:

Sets the output pin for Master In, Slave Out.

C Prototype:

```
void  SPIM_SetMISO(BYTE bPin)
```

Assembly:

```
mov   A, bPin
lcall  SPIM_SetMISO
```

Parameter:

BYTE bPin: 0x00--SPIM_MISO_P16, 0x01--SPIM_MISO_P15. It is passed in Accumulator.

Return Value:

None

Side Effects:

You can alter the A and X registers by this function.

SPIM_bIO

Description:

Transmits a single data byte and returns a received data byte from a slave device.

C Prototype:

```
BYTE  SPIM_bIO(BYTE bTXData)
```

Assembly:

```
mov   A, bRxData
lcall   SPIM_bIO
```

Parameters:

bTxData - data to transmit

Return Value:

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

Side Effects:

The status bits are cleared after this function is called. You can alter the A and X registers by this function.

Sample Firmware Source Code

The following sample code examples demonstrate SPI master transmitting and receiving data:

C Sample Code Example:

```
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
#define TIME_DELAY   {for(i=30000; i>0; i--);}
WORD i;

// The message will be rewritten by data received from SPI slave device
CHAR Message[] = "Hello SPI Slave";

void main(void)
{
    BYTE bIndex;
    SPIM_Start();
    SPIM_SetMOSI(SPIM_MOSI_P15); // Set MOSI pin to P1.5
    bIndex = 0;
    TIME_DELAY;
    while (bIndex < sizeof(Message))
    {
        // Send the whole message by loading the every next byte into TX buffer
        // Replace the transmitted byte with the received one from other SPI device
        Message[bIndex] = SPIM_bIO(Message[bIndex]);
        bIndex++;
        TIME_DELAY;  // Insert a user provided function here to make delay
    }
}
```

Assembly Sample Code Example:

```
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

macro TIME_DELAY
    mov   A, ffh
.timeloop:
    nop
    dec   A
    jnc   .timeloop
endm

AREA bss(RAM, REL)

bReadByte: BLK 1
export _main
export Message

AREA text(ROM, REL)

.LITERAL
Message:
    ASCIZ "Hello SPI Slave"
```

```
.ENDLITERAL

_main:
    lcall SPIM_Start
    mov   A, SPIM_MOSI_P15 ; Set MOSI pin to P1.5
    lcall SPIM_SetMOSI
    mov   X, 0             ; Set index to beginning of string

.TxNextByteLoop:
    mov   A, X
    index Message          ; Get the next symbol from the message string
    jz    .AllDone         ; Check whetеher all symbols were sent
    lcall SPIM_bIO         ; Load TX buffer with the next symbol
    mov   [bReadByte], A
    TIME_DELAY             ; Insert a user provided function here to make delay
    inc   X                ; Advance the pointer
    jmp   .TxNextByteLoop  ; and repeat until

.AllDone:
; Endless loop
    jmp   .AllDone
```

Version History

Version	Originator	Description
1.1	DHA	Added Version History.
1.20	DHA	Updated user module placement procedure to correct SPI pin values.
1.30	HPHA	1. Corrected method of clearing posted interrupts. 2. Updated SPI pin parameter value range for CY7C63803-LQXC.

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.