CE56273 – SPI With DMA in PSoC® 3 / PSoC 5

CE56273

Associated Part Families: CY8C38xx/CY8C55xx
Software: PSoC® Creator™
Related Hardware: CY8CKIT-001
Author: Anu M D

Objective

This code example demonstrates how to transmit data from a buffer through SPI using the DMA in PSoC® 3 / PSoC 5.

Overview

This code example explains how a master write and slave read is done using the DMA. The SPI transmission occurs on every switch press. RTC data stored in a memory buffer is transmitted through SPI using one DMA channel. Another DMA channel moves SPI data received by the slave onto a memory buffer. Both the master buffer and slave buffer data are displayed on the LCD.

This code example demonstrates the procedure of SPI data write using the DMA. For data read, the DMA configuration must be modified accordingly.

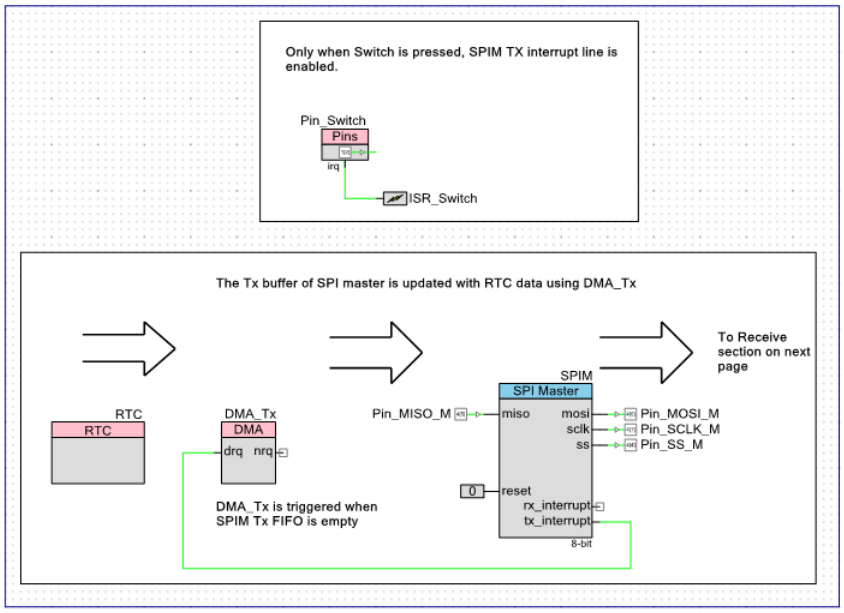
Component List

Instance Name	Component Name	Component Category	Comments
RTC	RTC	System	For the Real Time Clock (RTC) component to work, the 32 k XTAL should be enabled in the *cydwr*.
SPIM	SPI_Master	Communications	SPI master configured for 2 Mbps, Mode 00 transfer
SPIS	SPI_Slave	Communications	SPI slave having Mode 00 transfer
DMA_Tx	DMA	System	Default configuration
DMA_Rx	DMA	System	Default configuration
LCD	Character LCD	Display	Default configuration
ISR_Switch	Interrupt	System	Default configuration
Clock	Clock	System	A 4 MHz clock connected to the Clock input of SPI Slave that defines the sampling rate of the SPI slave status register
Pin_Switch	Digital Input Pin	Ports and pins	Configure this pin as input with resistive pull-up and enable interrupt on falling edge
SCLK_M	Digital Output Pin	Ports and pins	Default configuration
SCLK_S	Digital Output Pin	Ports and pins	Default configuration
MOSI_M	Digital Output Pin	Ports and pins	Default configuration
MOSI_S	Digital Input Pin	Ports and pins	Default configuration
MISO_M	Digital Input Pin	Ports and pins	Default configuration
MISO_S	Digital Output Pin	Ports and pins	Default configuration
SS_M	Digital Output Pin	Ports and pins	Default configuration
SS_S	Digital Input Pin	Ports and pins	Default configuration
ZeroTerminal_1	Logic Low	Digital → Logic	

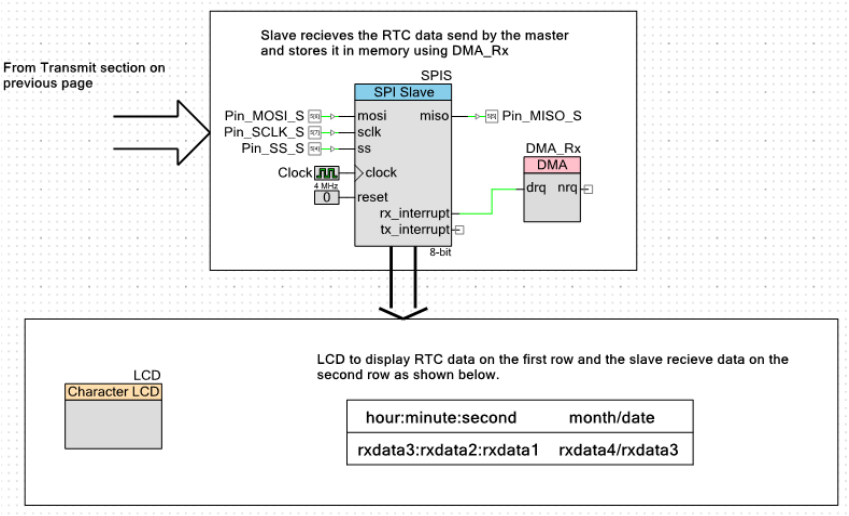
Top Design

The code example schematic is split across two sections – Transmit and Receive. The schematic for both of the sections is as follows.

Transmit



Receive

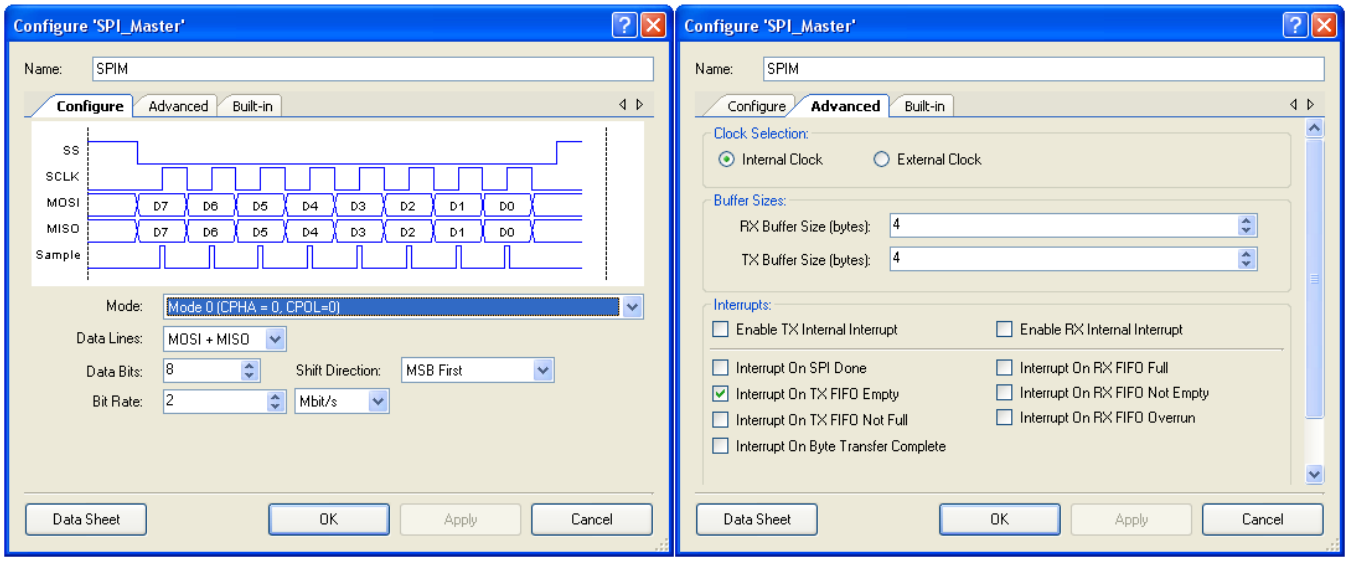


This pin configuration in the *.cydwr* file is as follows:

Alias	Name	Pin	Lock
	Pin_SCLK_S	P5[7]	☑
	Pin_MOSI_M	P4[6]	☑
	Pin_SS_S	P5[4]	☑
	Pin_MOSI_S	P5[6]	☑
	\LCD:LCDPort\[6:0]	P2[6:0]	☑
	Pin_Switch	P1[2]	☑
	Pin_MISO_M	P4[5]	☑
	Pin_SCLK_M	P4[7]	☑
	Pin_MISO_S	P5[5]	☑
	Pin_SS_M	P4[4]	☑

Component Configuration

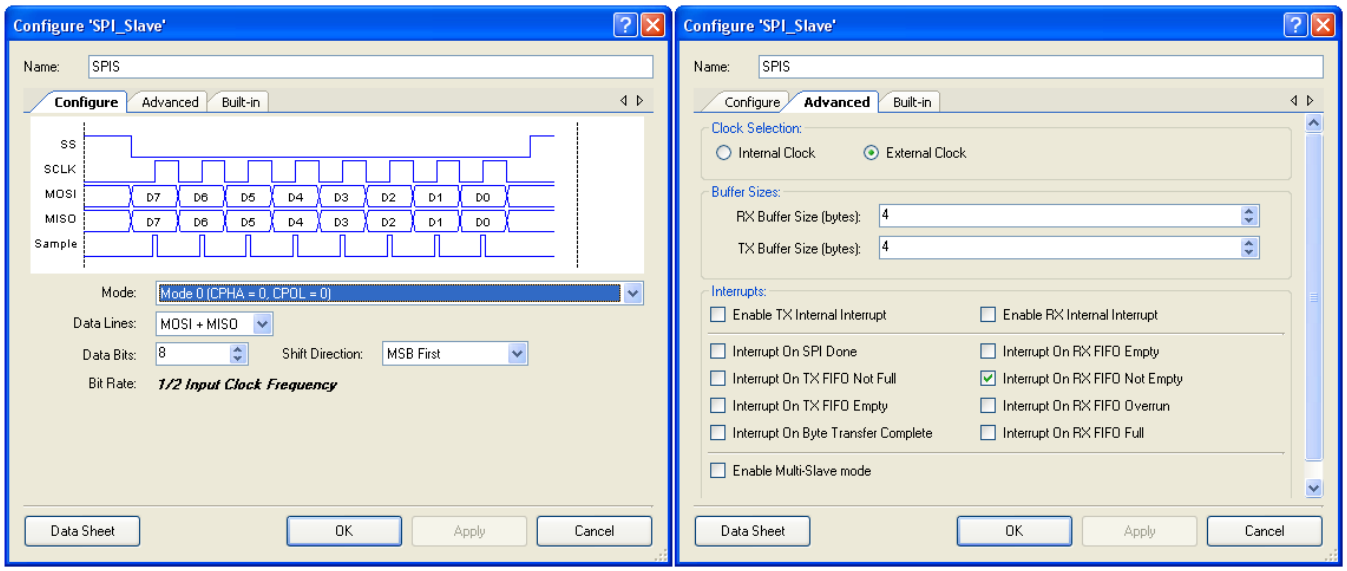
SPIM



The SPIM is configured to transfer at the rate of 2 Mbps with mode 0.

Enable Interrupt on Tx FIFO Empty so that the Interrupt line can be used as a trigger for a DMA_Tx transaction. RX and TX Buffer sizes are set to 4. That makes use of the FIFO buffer in Universal Digital Block (UDB), which is used for implementing SPI. If the buffer size is kept greater than 4, the buffer is implemented in SRAM. There is no need to enable Internal RX and TX Interrupts; those are required to pump data in and out of the SRAM buffer, which is created only when buffer size is greater than 4. This eliminates the CPU overhead during SPI data transfers.

SPIS



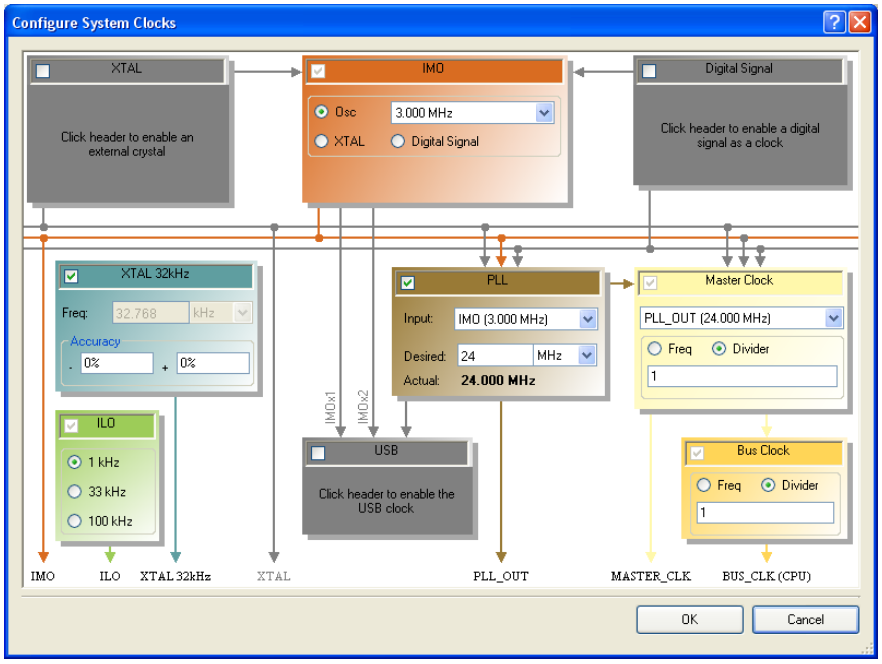
The SPI slave is configured for the same Mode 0, Data bits, and Shift Direction, as the SPI master. Interrupt on RX FIFO Not Empty is enabled so that the interrupt line goes high when a byte is received from the SPI master. This acts as the trigger for DMA_Rx to move data from the SPI slave Rx FIFO to the slaveRxData buffer.

Note: The SPIS component uses two clocks. The implementation is clocked from the SCLK input signal, and a second clock is used to report status information. The component properly handles the crossing between these two clock domains, but the static timing analyzer warns about signals crossing between these two clocks. It is safe to ignore all asynchronous clock warnings where the source clock is the SCLK pin. Do not ignore timing violations or other asynchronous clocking messages. The static timing analyzer will be improved in future releases to recognize these conditions and suppress the errant warnings.

Design Wide Resources

Clocks

To use the RTC component, you must enable the external crystal. Select the checkbox to enable the 32-kHz XTAL option in the clock tree.



The code example uses the default configuration for all other resources.

Operation

The aim of this code example is to transmit six bytes of RTC data containing information about seconds, minutes, hours, day of month, month, and year to the SPI master (SPIM) Tx buffer using a DMA (DMA_Tx). The DMA is triggered for each byte transfer using the SPIM interrupt line that is set active when the SPIM Tx FIFO is empty. As the DMA loads the data into the SPIM TX buffer, the interrupt line becomes inactive. When the TX buffer becomes empty after the SPIM data transfer, it activates the TX interrupt line again, triggering the DMA to transfer one more byte into the TX buffer. This repeats for six bytes. After six bytes transfers from SPIM, the next request to the DMA triggers a new transaction that disables the SPIM TX interrupt line, stopping further requests from the SPIM.

This interrupt is enabled again in the ISR: ISR_Switch is called on switch press. To disable and enable the SPIM TX interrupt line, the TX status mask register SPIM_TX_STATUS_MASK_REG is updated. Writing 1 into this register enables the interrupt and writing 0 disables the interrupt. Note that this interrupt line is connected to the DMA request line and it does not interrupt the CPU.

The SPI slave (SPIS) receives the RTC data. For each byte received, the DMA_Rx transfers the data from the SPIS to the memory buffer slaveRxData. This received data and the transmitted data are displayed on the LCD.

Configuring the DMAs

DMA_tx used on the SPIM side has two functions:

1. Transfer six bytes to SPIM TX buffer.
2. Disable the SPIM TX interrupt. This is done after six bytes are transferred.

To implement these two functions, two Transaction Descriptors are allocated. Each byte transfer is done on a SPIM TX interrupt.

The following table shows the APIs used to configure DMA_tx.

API	Configuration
DMA_Tx_DmaInitialize()	Burst size is set to 1 byte and to transfer each burst separate request is required
CyDmaTdAllocate()	Two TDs are allocated, ‘TD_tx’ to transfer six bytes and ‘TD_InterruptControl’ to disable the interrupt
CyDMATdSetAddress() for TD_tx	Source address is set to RTC data buffer ‘RTC_buffer’, and destination is set to ‘SPIM TX’ buffer. ‘SPIM_TXDATA_PTR’ is the pointer to the TX buffer.
CyDMATdSetAddress() for TD_InterruptControl	Source address is set to local variable ‘InterruptControl’ that holds data to disable ‘SPIM TX’ interrupt. Destination address set to ‘SPIM_TX_STATUS_MASK_REG’ that controls the ‘SPIM TX’ interrupt.
CyDmaTdSetConfiguration() for TD_tx	Transfer count is set to 6. Next TD is set to ‘TD_InterruptControl’ and source address is automatically incremented after each byte transfer.
CyDmaTdSetConfiguration() for TD_InterruptControl	Transfer count is set to 1. Next TD is set to ‘TD_tx’.
CyDmaChSetRequest()	CPU_TERM_CHAIN is passed as an argument to this API that clears the pending interrupts.
CyDmaChEnable()	Enables the DMA for executing transactions

DMA_Rx used on the SPIS side transfers the received bytes to SRAM array. A single TD is allocated to this DMA.

The following table shows the APIs used to configure DMA_rx.

API	Configuration
DMA_Tx_DmaInitialize()	Burst size is set to 1 byte and to transfer each burst separate request is required
CyDmaTdAllocate()	Single TD is allocated. Handle is stored in ‘TD_rx’
CyDMATdSetAddress() for TD_rx	Source address is set to ‘SPIS RX’ buffer and destination address is set to array ‘slaveRxData’
CyDmaTdSetConfiguration() for TD_rx	Transfer count is set to 6 bytes. Destination address is made to increment automatically for each byte transfer to point to next array location. Source address is same for all six bytes transfer.
CyDmaChEnable()	Enables the DMA_rx

For a detailed description of the DMA and APIs, refer to Using DMA in PSoC - AN52705. For details of the SPI master, slave components, and RTC, refer to the component datasheets.

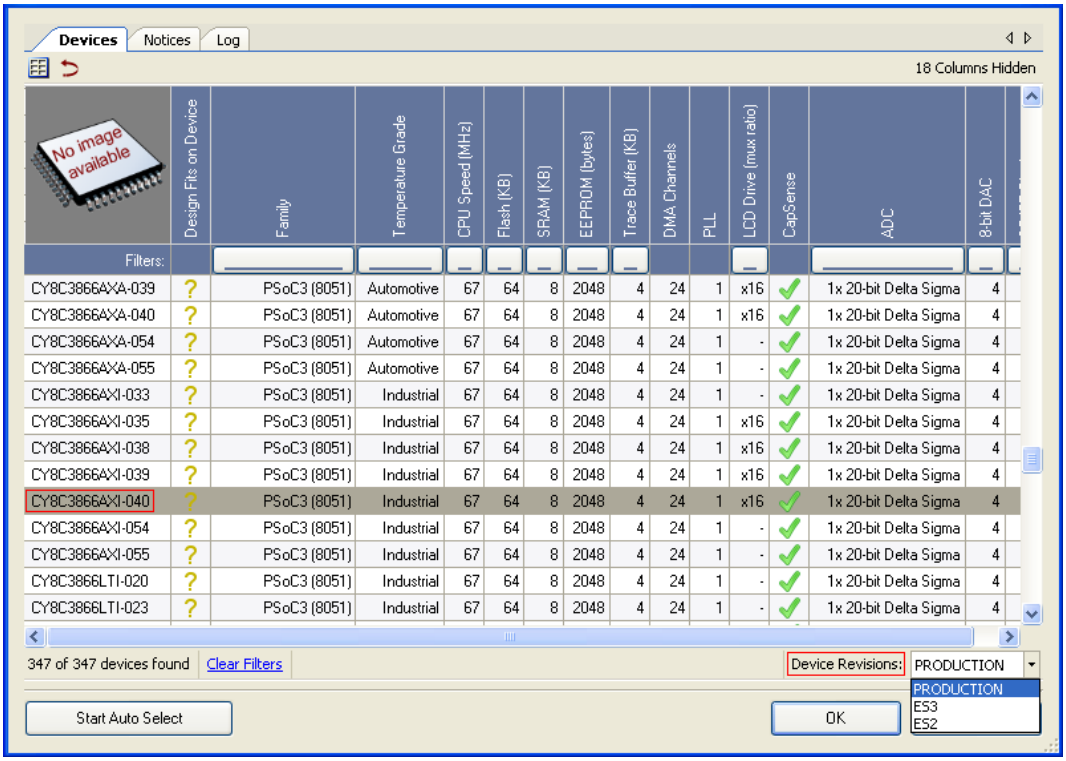
Hardware Connections

This code example can be tested on the CY8CKIT-001 development board. Make the following connections on the board to make the code example work:

- Connect P4[5] (SPI Master MISO) to P5[5] (SPI Slave MISO).
- Connect P4[6] (SPI Master MOSI) to P5[6] (SPI Slave MOSI).
- Connect P4[4] (SPI Master SS) to P5[4] (SPI Slave SS).
- Connect P4[7] (SPI Master SCLK) to P5[7] (SPI Slave SCLK).
- Connect P1[2] to switch SW1.
- Put the LCD module on P18. Refer to the PSoC Development Kit Board Guide for the jumper settings.
- Make sure that LCD power is turned on (J12).
- Make sure that your processor module has a 32-kHz crystal (in the backside of the PCB) so that the RTC will work.

Output

- Use the device selector window (Project → Device Selector) in PSoC Creator™ to select the appropriate device and device revision.
- If you are using a PSoC 3 device (for example, CY8C3866AXI-040) with production revision, then use the following selection.



Similarly, select the appropriate device number to work with the PSoC 5 Device family (for example, CY8C5588AXI-060).

Note For engineering samples, the device revision is marked on the package as part of the device number. Production silicon will not have an ES marking.

- Build the project and program the PSoC 3 device.
- Observe the LCD Display: The first row displays the RTC_buffer (Hr:Min:Sec Month/Date) data, and the second row displays the slaveRxData that is initially 0.
- Press SW1 to transmit RTC_buffer data to the SPI slave. Notice that the slaveRxData displayed on the second row of the LCD now has the same value as RTC_buffer when the switch is pressed.

  A sample LCD display output is as follows.

SW1 pressed

9 : 12 : 6	**5 / 25**
9 : 12 : 6	**5 / 25**

SW1 not pressed

9 : 12 : 8	**5 / 25**
9 : 12 : 6	**5 / 25**

Document History

Document Title: CE56273 – SPI With DMA in PSoC® 3 / PSoC 5

Document Number: 001-56273

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2766218	ANMD	09/24/2009	New spec
*A	2940628	ANMD	05/31/2010	Updated to PSoC Creator Beta 4.1 and made the projects PSoC 5 compatible.
*B	3012260	ANMD	08/26/2010	Updated component configuration and operation sections.
*C	3154535	ANMD	01/26/2011	Removed the Associated Project, Programming Language, and Prerequisites from the document header. Deleted the version number in the component table. Revised the output section for device selection.
*D	3197527	UDAY	03/16/2011	Replaced ‘example project’ with ‘code example’, and updated the title. Removed the Component Version column from the Components table. Updated to reflect the information related to STA warning. A note has been added to the SPIS component configuration section.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com/