# ELEMENTARY USAGE OF THE SERIAL PERIPHERAL INTERFACE COMPONENT FOR PSOC CREATOR

*Nathan Ward*
*November 11, 2011*
*Design Team 1*

## ABSTRACT

This application note goes over a primary component of the Programmable System on Chip (PSoC) communication library, the Serial Peripheral Interface (SPI) component. The note will first discuss various aspects and properties of the SPI block itself, and then demonstrate how the SPI block can be used in a basic application for the communication between two PSoC SPI blocks. The goal of this application note is to provide a simple framework from which a developer or first time user of PSoC can quickly understand and be able to use the SPI block within PSoC.

Keywords: SPI, PSoC, Schematic Editor, MOSI, MISO, Component Catalog

## BACKGROUND

The PSoC platform contains various configurable "blocks" which act as virtual electronic components. This capability makes PSoC an adaptable and powerful platform for a variety of applications. The blocks that can be utilized within PSoC Creator are diverse, ranging from digital blocks which can perform boolean logic or perform tasks such as multiplexing lines dynamically created by the user within PSoC Creator's Schematic Editor to analog capabilities such as digital-to-analog conversion and amplification of waveforms. However, beyond these digital and analog capabilities, there are blocks which are designed to make PSoC compatible with various communication protocols, one such block allows the platform to interface with external hardware via SPI.

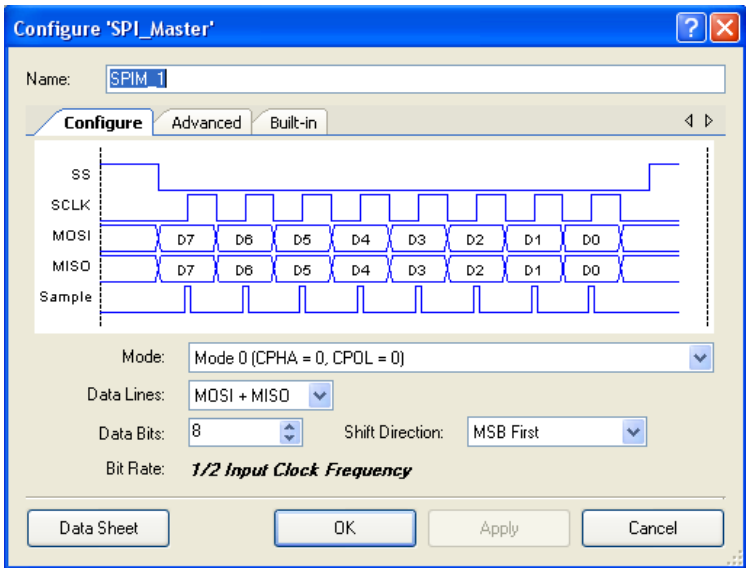



Figure 1: Configuration window of the SPI block

Blocks placed within the Schematic Editor can be modified in their behavior via their user interface, this functionality allows the developer to edit the properties of the successive blocks without having to change the various low-level

hardware libraries which characterize these virtual components. The SPI follows this trend and allows the user to modify how data is being transmitted through its individual user interface rather than through software coding, though this is also an option should the user choose to go this route. In Figure 1 we see some basic settings of the SPI which can be modified, namely the mode, data lines, data bits, shift direction, and bit rate. In mode, we set the CPHA and CPOL parameters of the signal which are listed, along with their respective effects, in Table 1. In the data lines parameter, we set whether we want the SPI to have two lines which are set to input and output, MISO and MOSI, or if we want a bidirectional port. Data bits specifies the bit-width of a single transfer which can be set from 3 bits to 16 bits. Shift direction specifies whether the data goes in with the most significant bit first or the least significant bit first, and the bit rate sets the clock speed. These are the basic settings which can be adjusted, however, there are many more aspects of the SPI which can be tweaked to suit a particular application.

Table 1:

| Mode | CPHA | CPOL | Description |
|---|---|---|---|
| 0 | 0 | 0 | Clock base value zero. Data captured on rising edge, propagated on falling edge. |
| 1 | 0 | 1 | Clock base value zero. Data captured on falling edge, propagated on rising edge. |
| 2 | 1 | 0 | Clock base value one. Data captured on falling edge, propagated on rising edge. |
| 3 | 1 | 1 | Clock base value one. Data captured on rising edge, propagated on falling edge. |

Under the advanced tab, the user can specify many more properties of the SPI block (Figure 2), which can be broken up into three categories: clock selection, buffer size, and interrupts. Clock selection determines whether the clock is integrated with the SPI block or if it is coming from an external source. In the example shown later in this paper, the clock signal is coming from an external timing block; since an external source is used, the bit rate setting as mentioned previously will not actually set the clock speed but rather will automatically divide the frequency of the incoming clock signal by two. The buffer size sets the number of bytes stored in the RX and TX buffer, which can be between four and 255, setting the buffer less than that will cause an error message to display. The third category in the advanced tab is the enabling of interrupts on certain events.

The interrupts will be sent out of the SPI block via the RX or TX interrupt output lines which can activate either custom or predefined interrupt service routines laid out in the Schematic Editor. Setting the RX or TX buffer above 4 bytes has certain effects on the interrupts, namely causing the "FIFO NOT EMPTY" interrupt to be permanently enabled to prevent incorrect buffer functionality. Users which require the use of interrupts or SPI functionality beyond the scope of this paper are encouraged to consult the SPI component data sheet provided by Cypress Semiconductor through double clicking on the SPI block and then pressing

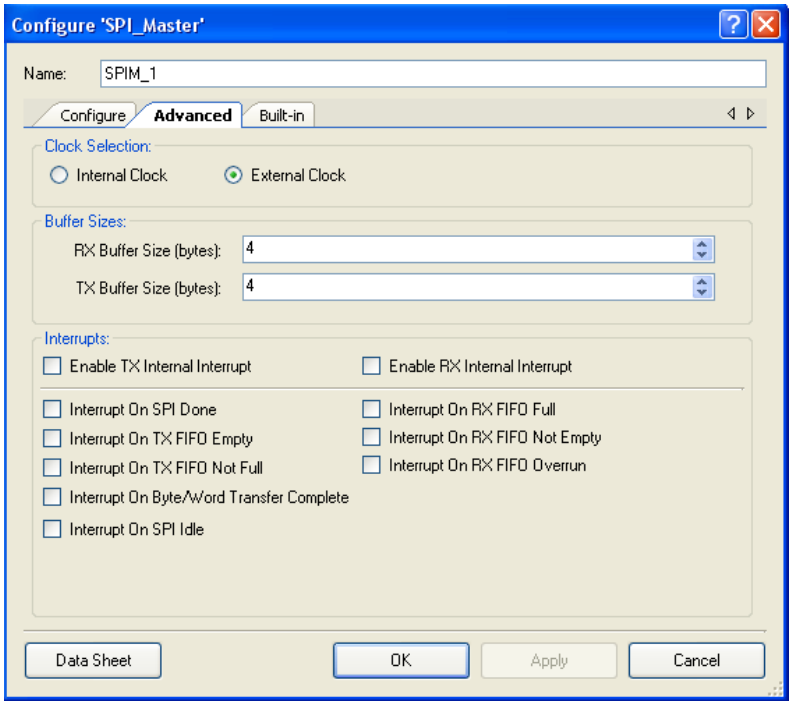



Figure 2: Advanced tab of the SPI block

the “Data Sheet” button.

## SPI EXAMPLE

In this example, two SPIs are being used to demonstrate both the functionality of the master and the slave SPI blocks. This example shows a virtual hardware configuration not typical of most PSoC projects. Whereas most of the time the PSoC is used as a component in a greater system, this demonstration will have the components be all internal to the PSOC without any input or output pins. However, this functionality is easily added through the "ports and pins" folder within the Component Catalog menu.

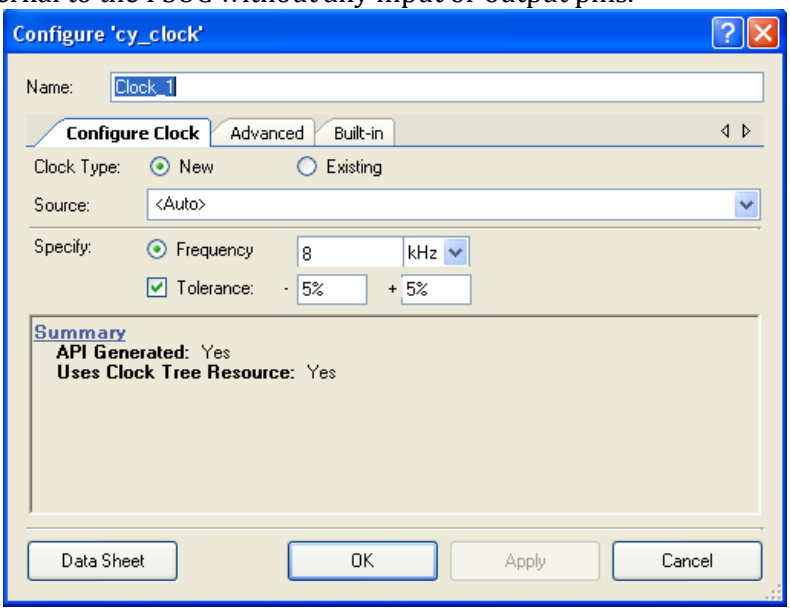



Figure 3: Configuration window of the clock block

To start this example, open up PSOC creator and create a new project. Once a new project has been generated, we can start laying down the blocks which make up the virtual hardware of our project. Open up the Schematic Editor for the project and also find the Component Catalog. The Component Catalog contains all of the blocks we will need for our project, namely the SPI slave and master blocks. Under the Component Catalog, search under the communication subfolder, and then under the SPI subfolder, this is where the SPI components will be. In this project we will be using SPI Master Block and the SPI Slave Block. Click and drag these components onto the Schematic Editor and place them an adequate distance apart; at this point, we now only need a couple of other virtual components before moving onto the software portion of this tutorial.

For this example, we will be using an external clock source. For many applications it is advisable to use an external clock source so that the design is as modular as possible, this also allows simpler integration of the SPI clock signal with other components. Find the system subfolder in the Component Catalog and drag two clock blocks onto the Schematic Editor, one will be for the SPI Master block and the other will be for the SPI Slave block. Although this step is not necessary, the clock speed can be adjusted to suit this application. Because there is not a high throughput of data being transceived by the SPI blocks, I set the clock speed to 8 KHz. This can be done by double clicking the clock blocks and then specifying the frequency (Figure 3).

The last component we will need is a source for the reset which grounds the virtual pin. Look under digital/logic subfolder and take out two logic low components, place these components next to the reset pin for the SPI Master and SPI Slave. Now, the final step for us to complete in the Schematic Editor is placing virtual traces to connect all of our components. First click the button labeled “Wire Tool” on the left hand corner of the schematic editor and then apply the wires to the schematic, connecting the clock and reset pins to their respective components, and also connecting the SPI blocks together in the manner shown in Section A of the Appendix. Once this task has been completed, press F5 or alternatively click

Debug under the Debug menu. This will compile our code and then upload our design to the PSoC, our code won't do anything for now, but with this step we have verified our code is bug free and also generated the source and header files associated with each component.

For now, let's take a look at the source file SPIM_1.c under Generated Source/SPIM_1 which contains the functions of the SPI we will be calling to in this project, namely the SPIM_1_Start and SPIM_1_WriteTxData. SPIM_1_Start is really just a combination of two other functions within this source file, the initialize function and the enable function, which simplifies getting our module running. SPIM_1_WriteTxData will output an 8-bit unsigned integer out of the MOSI to be read by our SPI Slave device. The complementary code to read this signal is in the SPIS_1.c source file under the Generated Source/SPIS_1 subfolder, specifically the function SPIS_1_ReadRxData.

## RUNNING THE APPLICATION

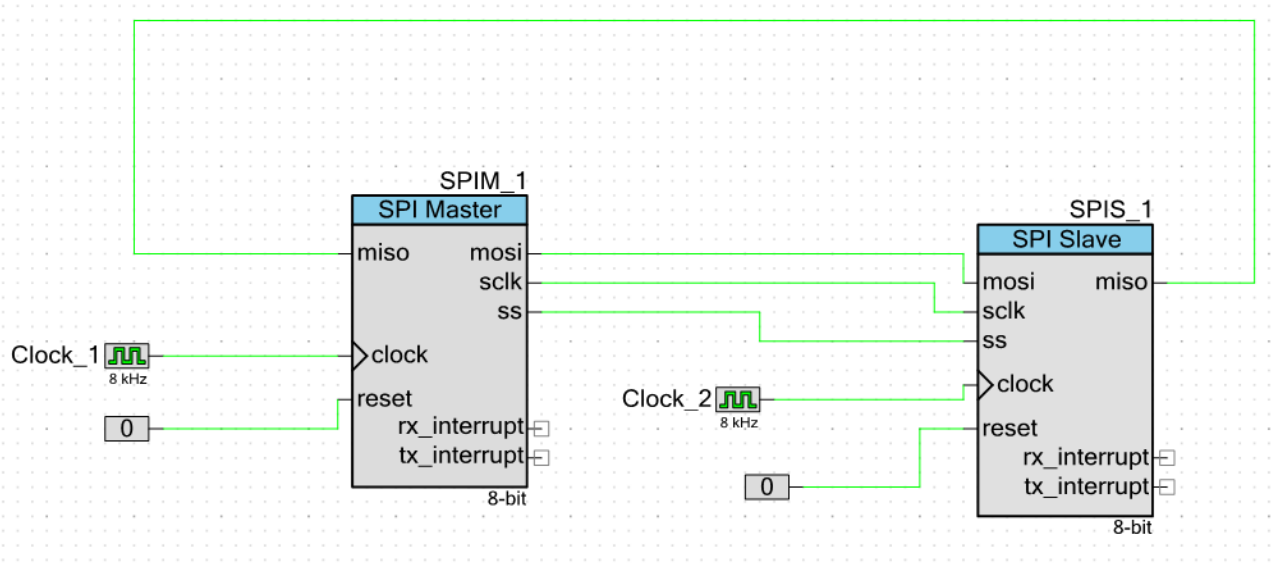
To actually make use of the code from the generated source and header files, first open Source Files/main.c. Make changes to the code according to the source listed below, once this is done, place a break point on the "SPIM_1_WriteTxData(5);" line and the closing bracket of main. Set the code to debug again and continue to parse the code until the first breakpoint. From here open up the locals tab on the bottom window of PSoC Creator, we can observe from this that our variable is still set to the value we initialized it to. Resume debugging (F5 or the "play" button in the toolbar) and observe that our local variable has changed to the value we read from the SPI Slave, the project is now working. Although this data is not doing anything objectively useful at the moment, in more advanced applications, the variable could be used for more complex purposes.

## CONCLUSION

In this application note, various properties of the SPI components were discussed. With the simple project demonstrated here, first-time users of PSoC can get projects involving the SPI communication protocol off the ground, and gain exposure to the communications module of PSoC. Additionally, this example can serve as a foundation for projects which utilize the full complexity and capability of the PSoC platform.

# APPENDIX

## SECTION A: SCHEMATIC EDITOR VIEW




## SECTION B: MAIN.C

```
/* ========================================
 *
 * Copyright Michigan State University, 2011
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF Michigan State University.
 *
 * ========================================
*/
#include <device.h>
#include <SPIM_1.h>
#include <SPIS_1.h>

void main()
{
      uint8 reader = 0;
      SPIM_1_Start();
      SPIS_1_Start();

      SPIM_1_WriteTxData(5); // place breakpoint here...
    reader = SPIS_1_ReadRxData();

} // ...and here

/* [] END OF FILE */
```