Application Note

Implementing SPI Master and Slave Functionality Using the Z8 Encore!® F083A

Z8 Encore!®

AN026701-0308

Abstract

This application note demonstrates a method of implementing the Serial Peripheral Interface (SPI) Master and Slave functionality on Z8 Encore! microcontrollers. The software implementation controls four GPIO port pins that are connected to the SCK, SS, MISO, and MOSI lines of the SPI bus.

Note: The source code (AN0267-SC01) associated with this application note is available for download from *www.zilog.com*.

Discussion

The Serial Peripheral Interface bus is a synchronous, serial communication link that operates in full duplex, meaning that a device transmits and receives data simultaneously. The devices communicate as a Master/Slave, where the Master initiates communication by selecting a Slave device with a hardware line and also provides the synchronous clock used to shift data bits in and out of the Slave. The signals required for communication are the Slave Select (SS), Master In Slave Out (MISO), Master Out Slave In (MOSI), and Serial Clock (SCK). The advantages of SPI over other communication protocols is that the addressing is performed in hardware with the SS line, making it faster to address a device, and that communication is full duplex, allowing for faster transfers of data. See Figure 1 on page 2.

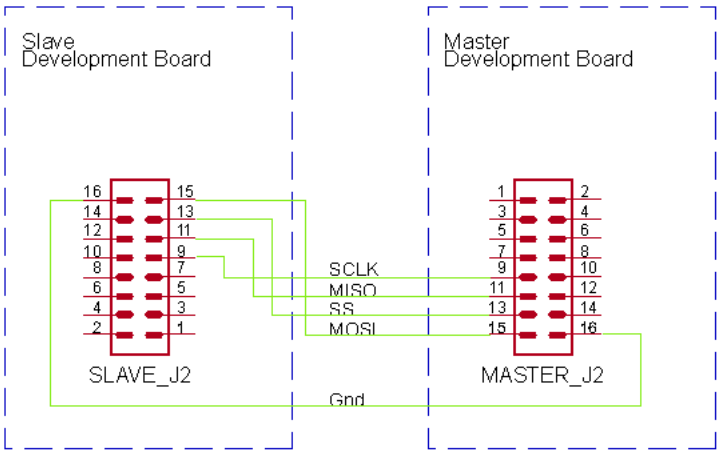

Figure 1. Serial Peripheral Interface Bus

SPI communication begins with the Master asserting the SS line. Depending upon the device, the SS line might be active high or active low. The Master must then wait at least one clock period before starting communication. Much like the active polarity of the SS line, the waiting period after SS activation varies from device to device. As an example, an analog-to-digital converter might require that the Master wait for a conversion to be completed after its SS line has been asserted. Next, the Master will begin shifting data out of the MOSI line, and it will shift data in on the MISO. Data is always transferred as full duplex, even when that data is not meaningful. As an example, for a Master to receive 24 bits of data from a Slave device, it must also transmit 24 bits to the Slave device. See Figure 2 and Figure 3.

Phase 0 Timing

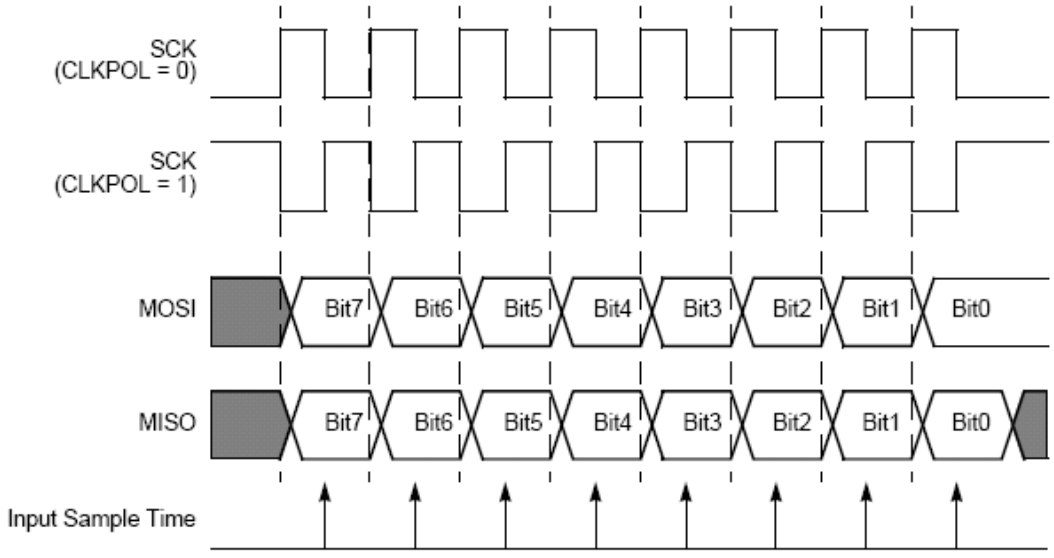

Figure 2. Phase 0 Timing

Phase 1 Timing

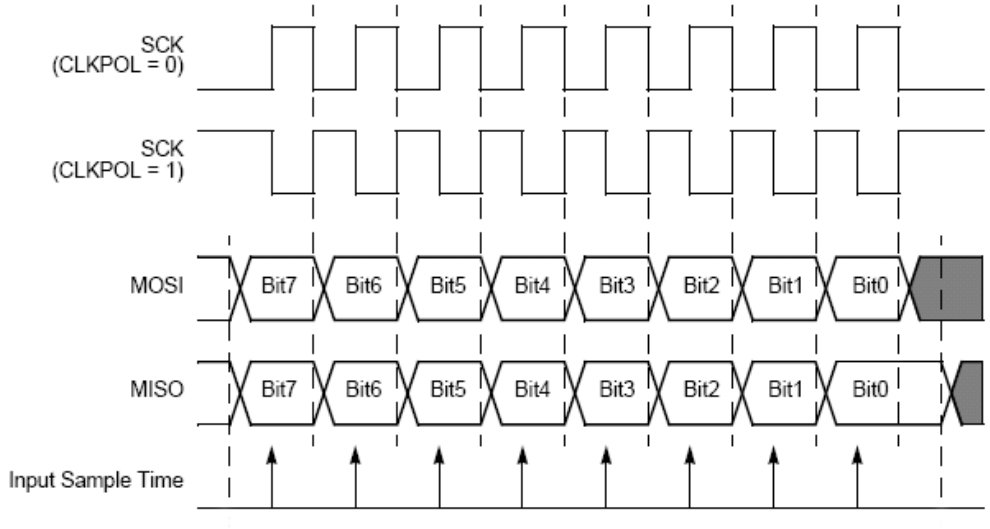

Figure 3. Phase 1 Timing

There is no standard for which clock edges are used for transmitting and receiving data, which leaves four possible modes of operation, depending on clock polarity and clock phasing. See the following table.

SPI Modes

MODE	SCK Polarity	SCK Phase	SCK Transmit Edge	SCK Receive Edge	SCK Idle State
0	0	0	Falling	Rising	Low
1	0	1	Rising	Falling	Low
2	1	0	Rising	Falling	High
3	1	1	Falling	Rising	High

Using Mode 1 as an example, the Master would idle the bus with the SCK line low. When the Master pushes the SCK line high, it will also place the most significant bit first on the MOSI line. At the same time, the Slave will place the most significant first on the MISO line. Next, the Master pulls the SCK line and reads the stable bit from the Slave on the MISO line. At the same time, the Slave reads a stable bit generated from the Master on the MOSI line. The communication is terminated when the SS line is deactivated, so it must remain active during the entire communication frame.

Developing Software Emulation

The Z8 Encore! F0830 and F083A microcontrollers do not have a hardware peripheral for SPI support; however, the protocol is very easy to implement in software.

Master

The software from the Master side will require generating the SS, SCK, and MOSI lines, while reading data on the MISO line. The Master first activates the SS line and then waits one clock period for the Slave to prepare for communication. Next, the Master toggles the SCK line and shifts data in or out depending on the polarity of the SCK line. Finally, the Master waits one clock period before repeating the process for all 8 bits. This process is repeated until the Master's buffer is emptied, at which point the communication is completed, and the Master deactivates the SS line.

Slave

The software on the Slave side is slightly more complicated because the Slave will need to respond quickly to communication requests from the Master. One hardware interrupt will be used to detect the SS line. When the Slave Select ISR has been entered, the Slave will monitor transitions on the SCK line by polling a second hardware interrupt. When an edge has occurred, the Slave will then shift data in or out depending on which phase of the SCK line has been detected. The Slave will continue the process of monitoring SCK transitions and shifting bits until the ISR is terminated by the Master deactivating the SS line.

Software

Master

The definitions required for building the Master driver are located in the spi_master.h file. These definitions configure the size of the input and out buffers, the port pins used for the bus, and the SPI mode used for clock phase and polarity.

Definitions

- DATASIZE
  Determines the size of the input buffer.
- SPIMode
  Determines which SPI mode is used for communication as detailed in the table on page 3.
- SCLKbit
  Determines which PortC bit is used for the SCK line. For example, if port pin PC5 is used, this would be set to 16.
- MISObit
  Determines which PortC bit is used for the MISO line.
- MOSIbit
  Determines which PortB bit is used for the MOSI line.

Globals

One global is used for the input buffer, **DataIn[]**.

APIs

- void SPI_Init().
  This function will set the bus lines in the correct direction and idle the SCK with the correct polarity.
- char SPI_SendReceive(char data)
  This function will transmit the byte in data and will return the byte received.
- void SPI_SendReceiveBlock(char* p_out, char* p_in, unsigned char length)
  This function will transmit length bytes (0 = 256 bytes) pointed by p_out and will store length bytes received in the buffer pointed by p_in.
- void WaitHalfBit()
  This function will wait one half clock period.

Usage

Configure all the definitions for your application and then build the driver with Zilog Developer Studio (ZDS) II. To perform a data transfer in your application, use the SPI_SendReceive function for a single byte or SPI_SendReceiveBlock with the parameter p_out pointing to the n data (n is passed in the parameter length) to send and the parameter p_in pointing to the buffer where the data received will be stored.

Slave

Definitions

- DATASIZE
  Determines the size of the input buffer. Because the Master determines how much data is exchanged on a transfer, DATASIZE needs to be at least as large as the number of bytes exchanged by the Master.
- SPIMode
  Determines which SPI mode is used for communication as detailed in the table on page 3.
- SCLKbit
  Determines which PortC bit is used for the SCK line. For example, if port pin PC5 is used, this would be set to 16. Because this driver uses a polled interrupt to monitor the state of SCK, this signal must remain on a pin that supports a hardware interrupt on both edges.
- MISObit
  Determines which PortC bit is used for the MISO line.
- MOSIbit
  Determines which PortB bit is used for the MOSI line.
- SSELbit
  The PC bit is used for the SS line. This pin must be a hardware interrupt pin that triggers an interrupt on the same edge as your application's SS line. For instance, if your application uses an active high SS line, then the pin used for the SS line must trigger a hardware interrupt on the rising edge.
- SSELirq
  The ZDS II name for the interrupt used on the SS line.

Globals

One global is used for the input buffer, **DataIn[]**.The following globals must not be used nor changed by the user application:

- **DataReady**
- **pOut**
- **pBaseOut**
- **pIn**
- **pBaseIn**
- **ByteBlock**

APIs

- void SPI_Init()
  This function will prepare the interrupt driven SPI Slave driver.
- char SPI_SendReceive(char data)
  This function will send a byte and return the byte received.
- void SPI_SendReceiveBlock(char* p_out, char* p_in, unsigned char length)
  This function will transmit length bytes (0 = 256 bytes) pointed to by p_out and will store length bytes received in the buffer pointed to by p_in.

Usage

Configure all the definitions for your application and then build the driver with ZDS II. Call SPI_Init() to configure the SPI driver and then enable interrupts when your application is ready for processing. When the Slave detects the SS line being asserted, the ISR SSELISR will be serviced. After all variables are initialized, the SCK line interrupt flag is pushed down so that any new transitions will be seen. The ISR then waits for a new transition on the SCK line. When a transition has been detected, the ISR verifies that the SSEL line is still asserted. If the SSEL line is not asserted, the exchange is considered complete; if the line is still asserted, the ISR will continue. Next, the ISR determines if the SCK line is in the transmitting or receiving phase and shifts a bit in or out accordingly. This process will continue until the Master deactivates the SS line.

Testing the SPI Master/Slave

Sample Hardware

Required Equipment

Two Z8 Encore! F083A 28-Pin Development Kits (see Figure 4 on page 7)

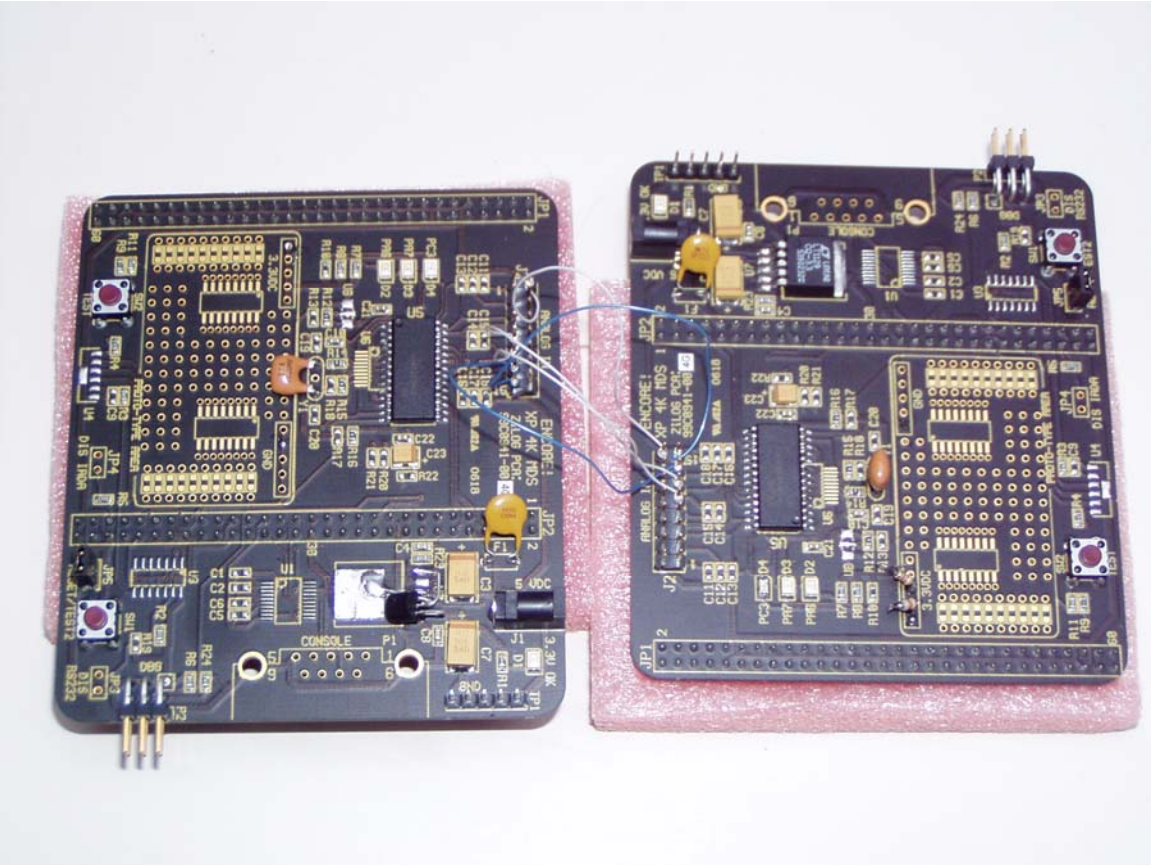

Figure 4. Two Z8 Encore! F083A 28-Pin Development Kits

Hardware Preparation

One development kit will be used as a Master, and the second development kit will be used as a Slave device. Along with a common ground, the SS, SCK, MOSI, and MISO lines will need to be jumpered together between the two devices.

1. Run a wire from the Master's J2-16 to the Slave's J2-16. This will serve as ground.
2. Run a wire from the Master's J2-9 to the Slave's J2-9. This will serve as the SCK line.
3. Run a wire from the Master's J2-11 to the Slave's J2-11. This will serve as the MISO line.
4. Run a wire from the Master's J2-13 to the Slave's J2-13. This will serve as the SS line
5. Run a wire from the Master's J2-15 to the Slave's J2-15. This will serve as the MOSI line.
6. Build the Master ZDS project and flash the code into the Master development board.
7. Build the Slave ZDS project and flash the code into the Slave development board.
8. Reset the Slave and the Master development boards by pressing SW1 or power cycling the boards.

Sample Software

Master

The Master software will send the string “Hello ZiLOG!” to the Slave; then it will wait until the switch SW2 is pressed. The Slave will respond with the same string to the Master and will light D3 if the string received is correct. If this response is not received by the Master, then the Master will light LED D4 to signal that an error has occurred. If no error occurs, the Master lights LED D2.

Then, with each press of switch SW2, the Master will increase the value of the byte sent to the Slave until the value reaches 128, and then the Master will start over again sending a value of 1. If the response sent by the slave is equal to 0xAA, the Master will light LED D2. If it is not, the Master will light LED D4.

At power-up, the Master will call SPI_Init() to configure the SCK, SS, MISO and MOSI lines for the correct direction and idle state. Next, the Master calls IdlePorts() to configure switch SW2 and LEDs D2, D4 for the correct direction.

The Main Loop waits 62.5 mS to help debounce the switch, and then the switch is read. If the switch is closed, a byte is increased for use by the Slave’s PWM. Next, the Master starts the exchange by calling SPI_SendReceive(). After SPI_SendReceive() completes, the Master compares the data received to verify that the Slave responded 0xAA. If the Slave responded, then LED D4 is turned off, and LED D2 is turned on. If the Slave did not respond, then LED D2 is turned off, and LED D4 is turned on.

Slave

The Slave software will wait for an exchange from the Master and perform a PWM on LED D3 based upon the value received by the Master.

At power-up, the Slave will call SPI_Init() to configure the SCK, SS, MISO and MOSI lines for the correct direction and idle state. Next, the Slave will call IdlePorts() to initialize the PWM on LED D3. Now, the Slave will call SPI_SendReceive() for data to be exchanged from the Master by polling the state of DataReady. If the data has been received, then the Slave will update the value of the PWM. Throughput can be measured using an oscilloscope to monitor the SCK line during an exchange.

Summary

This application note demonstrates a method of implementing the Serial Peripheral Interface Master and Slave functionality on Z8 Encore! microcontrollers. The software implementation controls four GPIO port pins that are connected to the SCK, MISO, SS, and MOSI lines of the SPI bus.

Appendix A—References

Topic	Document Name
Z8 Encore!® MCU	*Z8 Encore! F083A Series Product Specification* (PS0263)
	Z8 Encore! MCU User Manual (UM0128)
ZDS II	*Zilog Developer Studio II—Z8 Encore! User Manual* (UM0130)
SPI Master	*Z8 Encore! XP F64xx Series Product Specification* (PS0199)
Development Kit	*Z8 Encore! F083A Series Development Kit User Manual* (UM0206)
SPI Protocol	http://www.embedded.com/story/OEG20020124S0116

Warning: DO NOT USE IN LIFE SUPPORT

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer