Serial Peripheral Interface (SPI) and Microwire/Plus implementation Using the SX Microcontroller

Application Note 20
September 1999

1.0 Introduction

Synchronous serial interfaces are widely used to provide economical board-level interface between different devices such as microcontrollers, DACs, ADCs and other. Although there is no single standard for the synchronous serial bus, there are industry accepted guidelines based on the two most popular implementations: SPI (a trademark of Motorola Semiconductor) and Microwire/Plus (a trademark of National Semiconductor). Many IC manufacturers produce components that are compatible with SPI and Microwire/Plus. This documentation describes the operation of SPI and details how it can be implemented on a SX microcontroller by the use of two SPI Virtual Peripheral™ software modules.

2.0 SPI Description

SPI uses a master-slave model and typically has three signal lines: data input line, data output line and clock line. Chip select signals from the master are used to address different slaves on the bus (Figure 2-1). The hardware realization of such an interface is a simple shift register. The data bits are shifted in/out MSB (most significant bit) first. Often the data is shifted simultaneously out from the output pin and into the input pin. SPI interface defines only the communication lines and the clock edge, other parameters vary for different devices. Clock frequencies happen to be anywhere from 100kHz to a few MHz and word lengths are from 8 to 16 or more bits.

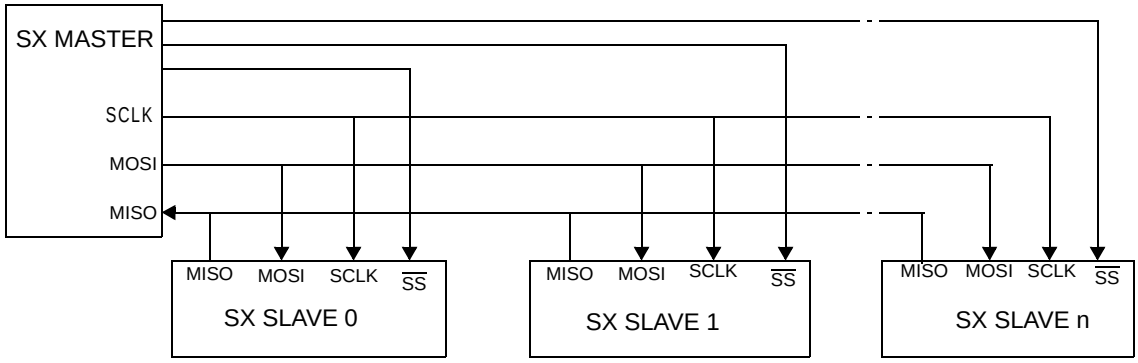

Figure 2-1. Master-Slave System Configuration

2.1 SIGNAL DESCRIPTIONS

The four basic signals (MOSI, MISO, SCLK and $\overline{SS}$) are described in the following paragraphs.

2.2 MASTER OUT, SLAVE IN

The MOSI line is defined as an output line from the master device to an input into the slave devices. The MOSI line transfers data in one direction only, from the master to a slave.

2.3 MASTER IN, SLAVE OUT

The MISO line is defined as an input line into the master device and as an output in a slave device. The MISO line transfers data in the opposite direction to the MOSI line, it transfers data from a slave device to the master.

2.4 SERIAL CLOCK

The SCLK line is used to synchronize both data in and out of a device via the MOSI and MISO lines. The SCLK line is generated by the master device and thus is an input into all slave devices.

2.5 SLAVE SELECT

The Slave Select ($\overline{SS}$) lines are controlled by the master to select a slave device. The $\overline{SS}$ line must be low prior to data transactions and must stay low for the duration of the transaction. Each slave device requires its own $\overline{SS}$ input line from the master.

The timing diagram of these lines can be seen in Figure 3-1.

3.0 SPI Operation

To initiate the data transfer between a master and slave device, the $\overline{SS}$ line must go low. This synchronizes the slave device with the master. Data can now be transferred between the master and slave device in one of two modes: either data is sampled on the rising edge of the clock or the falling edge of the clock. Figure 3-1 shows the data/clock timing diagram.

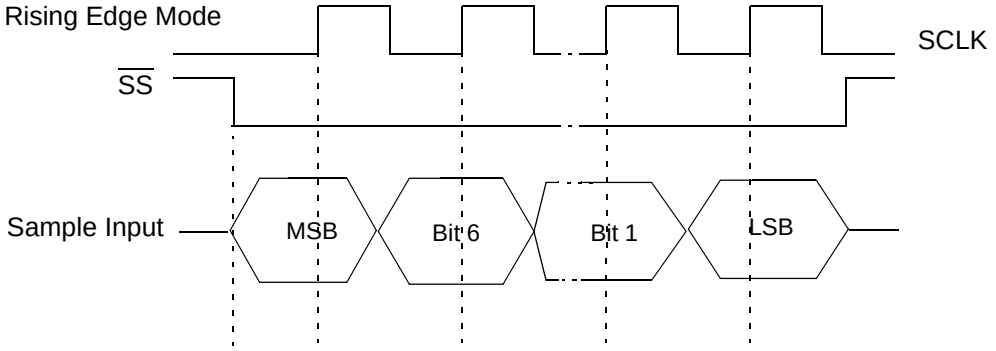

Figure 3-1. Data/Clock Timing Diagram

In a slave device a logic low is received on the $\overline{SS}$ line and the clock input is at the SCLK pin. This synchronizes the slave with the master. Data is then received serially at the MOSI pin. During a write cycle, data is shifted out onto the MISO pin on clocks from the master device. Figure 3-2 illustrates the signal line interconnections.

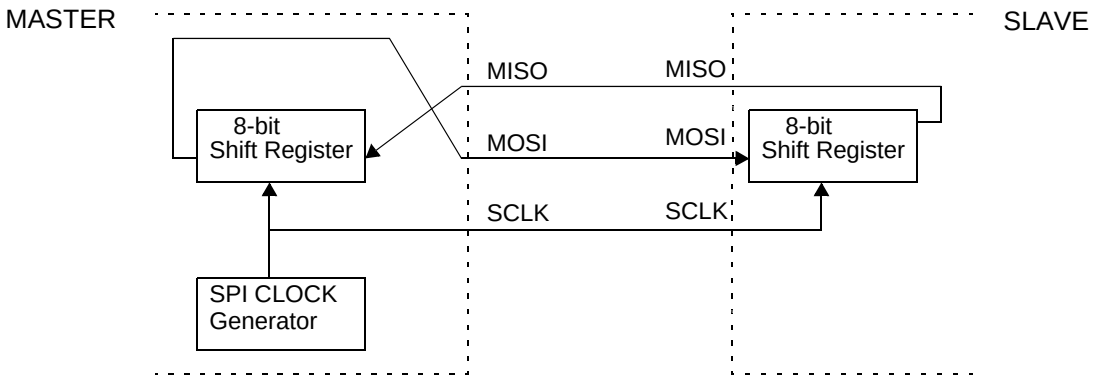

Figure 3-2. SPI Master Slave Interconnections

4.0 SX Implementation

Using the SX device, the SPI master-slave interface can be implemented using two Virtual Peripheral software modules, one for the slave devices and one for the master device. An SPI demo program that will be described later in this document, has been developed to demonstrate the effective use of these Virtual Peripheral modules.

Features:

- Full duplex, three-wire synchronous transfers
- Clock rate selectable up to 1.72MHz (2.5MHz possible with slight modifications)
- Master send frequency of 1.72MHz (max)
- Slave receive frequency of 1.1MHz (max)
- Word length selectable anywhere from 1 – 16bits. (More than 16bits easily implemented if required)
- Clock polarity is configurable by using the appropriate command byte
- Arbitrary pins of the SX52BD and SX18/SX20/28AC devices may be assigned for the SPI interface lines

Four I/O signal lines are required with any SPI communication:

Master-Out-Slave-In	MOSI signal line
Master-In-Slave_Out	MISO signal line
Serial Clock	SCLK signal line
Slave Select	$\overline{SS}$ signal line

4.1 SPIM (SPI MASTER) VIRTUAL PERIPHERAL (SPIM.SRC)

This Virtual Peripheral provides a simple and efficient way to send/receive data between SPI compatible slave devices.

4.2 SPIM CLOCK

SPIM Clock rate can be selected starting from 1.72 MHz (max) which corresponds to a minimum clock period of 580ns. Clock period can be incremented with 160ns increments. The maximum clock rate of 2.5MHz can be achieved by removing part of the program, responsible for the programmable clock rate. The minimum clock rate is practically unlimited. However, for clock rates lower than 500KHz, it is recommend that SPIML (low rate SPI Master Virtual Peripheral) be used. Clock polarity is configurable by using the appropriate command byte.

4.3 SPIM I/O PINS

SPIM controls the MOSI, MISO and SCLK lines. The user must assign the appropriate control for $\overline{SS}$ lines. Arbitrary pins of the SX device can be assigned to the signal lines used by SPIM.

4.4 SPIM DATA

Word length can be set to anywhere from 1 to 16 bits. Longer word lengths can be easily implemented if needed. If necessary, the user may want to add Read/Write and handshaking signals. This can be easily implemented without affecting the core SPIM program.

4.5 USING SPIM

To initialize SPIM, the user program must perform the following steps:

- Disable the interrupts
- Initialize SX I/O ports (calling SPIM_INIT)
- Place the outgoing data into the I/O buffer
- Set $\overline{SS}$ signal low and call SPIM_EXECUTE

Since SPIM is not using any interrupts, it will transfer the data and return control to user program.

The two-byte send/receive buffer is allocated in memory as SPIM_LSB and SPIM_MSB. Data is shifted starting from the most significant bit (MSB). In case of less than 16-bit long data words, the SPIM will automatically align the MSB of the word to the MSB of the buffer before shifting. No user intervention is required. It is important to note that the contents of SPIM_LSB and SPIM_MSB are not preserved during the driver operation since the bits are always shifted in.

4.6 SPIM CONTROL, STATUS AND I/O DATA REGISTERS

SPIM_RATE. This byte defines the clock rate. When set to 1, the maximum clock rate is 1.72 MHz (580ns period). The actual clock period can be calculated as:

Clock Period = 420 ns + (160 ns x SPIM_RATE)

If the contents of SPIM_RATE is zero, the minimum clock rate of 24.17kHz (41.380 us Period) is selected.

SPIM_PORT. This register must be configured to point to the SX Ports (A, B, C, D, or E) which will generate the SPI Clock signal. For instance:

```
spim_port     =      rb
```

SPIM_PORTA_MASK, SPIM_PORTB_MASK,SPIM_PORTC_MASK.These bytes are the images of SX I/O port direction registers. The values written into these registers correspond to the ports direction registers. A value of “1” defines the pin as an input while a value of “0” defines the pin as an output. For the master device, all the signal lines must be defined as outputs except the MISO line.

SPIM_CLK_MASK defines the SPI Clock pin. The pin is defined by setting the corresponding bit of SPIM_CLK_MASK. For instance, to define RC1 as the SPI clock input pin (SCLK):

```
spim_portc_mask          =      $fd
spim_port                =      rc
spim_clk_mask            =      $02
```

SPIM_CMD This register is defined as the command and configuration byte. This byte defines:

- SPI Clock polarity
- Data direction (Read/Write)
- Number of bits to send/receive.

SPIM supports 4 commands, specified by the 4 MSB bits of the SPIM_CMD byte.

They are defined as:

4 MSB Bits	Command
0011	GET WORD in MASTER FAST mode, define FALLING edge;
0111	GET WORD in MASTER FAST mode, define RISING edge;
1011	SEND WORD in MASTER FAST mode, define FALLING edge;
1111	SEND WORD in MASTER FAST mode, define RISING edge.

Practically there is no difference between the GET and SEND commands. When the I/O buffer is shifted out, the input data is always shifted in. The different commands are provided for user convenience and to avoid confusion.

The four LSB bits of the command byte define the word length.

4 MSB Bits	Command
0000	Sets the length to 16 bits
0001	Sets the length to 1 bit
…..	……..
1111	Sets the length to 15 bits

The SPIM_CMD byte also serves as a ready flag. SPIM will clear this byte upon the completion of data transfer.

The SPIM_STATUS byte provides the error status information. It has 1 error flag. Bit 4 is set to 1 when the SPIS_CMD byte contains an unsupported command.

4.7 SPIM ENTRY POINTS

SPIM_INIT. Initialization subroutine. It provides appropriate configuration for the SX I/O pins.

SPIM_EXECUTE. This is the actual Virtual Peripheral entry point, which may be called either from the interrupt service routine or from MAIN program.

SPIM_SS_SET. User defined subroutine to set active level on SS pin.

SPIM_SS_RESET. User defined subroutine to set passive level on SS pin. In the case of multiple devices on the SPI bus, the user must provide appropriate control over the SS signals.

SPIM_ORG. This is the base address of the SPIM Virtual Peripheral. Typically it will be the next program memory location, after the user program.

4.8 USING SPIM FROM THE USER PROGRAM

1. Make sure that no interrupt is allowed during execution of SPIM.
2. Verify that the execution parameters are configured properly:
   - SPIM_RATE
   - SPIM_PORT_A_MASK
   - SPIM_PORT_B_MASK
   - SPIM_PORT_C_MASK
   - SPIM_PORT
   - SPIM_CLK_MASK
   - SPIM_CMD
3. Configure I/O ports by calling SPIM_INIT.
4. Verify that SPIM is ready for execution by testing the SPIM_CMD byte.
5. Place the outgoing data into SPIM_LSB and SPIM_MSB.
6. Call SPIM_EXECUTE from the interrupt.
7. Test SPIM_CMD for completion and SPIM_STATUS for errors.
8. Unload the data from SPIM_LSB and SPIM_MSB.

SPIM is now ready for reuse. Repeat from step 4.

5.0 SPIS (SPI Slave) Virtual Peripheral Description (SPIS.SRC)

This Virtual Peripheral provides a simple and efficient way to send/get one word of data to/from a SPI compatible MASTER device.

5.1 SPIS CLOCK

SPIS is developed to work with relatively fast SPI devices. It occupies the CPU during the entire data transfer. Maximum clock rate is 1.1 MHz (900 ns clock period). The clock rate may vary during data transfer cycle. If the watchdog timer is enabled, this will limit the minimum clock rate.

Clock polarity is configurable by using the appropriate command byte.

5.2 SPIS DATA

Word length can be set to anywhere from 1 to 16 bits. Longer word lengths could be easily implemented if needed. SPIS does not make any differentiation between read and write cycles. The data is clocked in and out simultaneously.

5.3 SPIS I/O PINS

SPIS makes use of the Wake-UP interrupt from the SS signal. Therefore, the SS signal must be assigned to one of the Port B pins. Other signals may use any other free pins.

5.4 USING SPIS

To initialize SPIS, the user program must disable the RTCC interrupt and enable the appropriate edge (wakeup interrupt from the SS pin). SPIS will then wait for the interrupt from the SS input. When the interrupt occurs, SPIS will start execution of the send/receive loop, waiting for SPI clock signal and start processing the incoming and outgoing data lines. After the predefined number of clock pulses, SPIS will finish the communication cycle and exit the interrupt routine. In order to prevent locking of the driver in case of a communication error, SPIS uses it's internal SPIS_WATCHDOG counter, which would expire if SPIS has not received the clock pulse for too long, the data transfer will be aborted.

The two-byte (16 bits) send/receive buffer is allocated in memory as SPIS_LSB and SPIS_MSB. Data is shifted starting from the most significant bit. In case of less than 16-bit long data word, the SPIS will automatically align the MSB of the word to the MSB of the buffer before shifting (no user intervention is required). The contents of SPIS_LSB and SPIS_MSB are not preserved during the driver operation since the bits are always shifted in.

5.5 SPIS CONTROL, STATUS AND I/O DATA REGISTERS

SPIS_WATCHDOG. This byte defines the timeout period for the SPI slave. If SPIS will not receive the clock signal within this waiting period, the communication will be aborted and the error flag will be set. The actual timeout can be calculated as 160ns multiplied by SPIS_WATCHDOG.

SPIS_PORTA_MASK,SPIS_PORTB_MASK, SPIS_PORTC_MASK. These bytes are the images of SX I/O ports direction registers. They are written to corresponding ports direction registers during SPIS initialization.

SPIS_WKUP_MASK - SPIS writes SPIS_WKUP_MASK into Port B Wakeup Enable register.

SPIS_CMD - Command and configuration byte. This byte defines:

- SPI Clock polarity
- direction (Read/Write)
- number of bits to send/receive

SPIS supports 4 commands, specified by the 4 MSB bits of the SPIS_CMD byte:

4 MSB Bits	Command
0001	GET BYTE in SLAVE mode, define FALLING edge
0101	GET BYTE in SLAVE mode, define RISING edge
1001	SEND BYTE in SLAVE mode, define FALLING edge
1101	SEND BYTE in SLAVE mode, define RISING edge

The contents of SPIS_LSB and SPIS_MSB are not preserved during the r operation since the bits are always shifted in even if they don't make any sense.

The four LSB bits of the command byte define the word length.

4 MSB Bits	Command
0000	Sets the length to 16 bits
0001	Sets the length to 1 bit
…..	…. ...
1111	Sets the length to 15 bits

The SPIS_CMD byte also serves as a ready flag. SPIS will clear this byte upon the success of data transfer.

SPIS_STATUS. Provides the error status information to the user program. It has 2 error flags:

bit 4 - set to 1 when the SPIS_CMD byte contains an unsupported command

bit 7 - set to 1 when the data transfer was not completed successfully

5.6 SPIS ENTRY POINTS

SPIS_INIT. Initialization subroutine, it provides appropriate configuration for SX I/O pins.

SPIS_EXECUTE. This is the actual VP entry point, which is normally called from the interrupt service routine.

SPIS_ORG. This is the base address of SPIM VP. Typically it will be next ROM location, after the user program.

5.7 USING SPIS FROM THE USER PROGRAM

1. Verify that the execution parameters are configured properly:
   - SPIS_WATCHDOG
   - SPIS_PORT_A_MASK
   - SPIS_PORT_B_MASK
   - SPIS_PORT_C_MASK
   - SPIS_WKUP_MASK
   - SPIS_CMD
2. Call SPIS_INIT
3. Verify that SPIS is ready for execution by testing the SPIS_CMD byte.
4. Place the outgoing data into SPIS_LSB and SPIS_MSB
5. Disable the RTCC interrupt and enable the SS interrupt from Port B.
6. Call SPIS_EXECUTE from the interrupt.
7. Test SPIS_CMD for completion and SPIS_STATUS for errors.
8. Unload the data from SPIS_LSB and SPIS_MSB

SPIS is now ready for reuse. Repeat from step 3.

6.0 Demo Description

SPI DEMO provides an example of data transfer between two SX microcontrollers through the SPI data bus. SPI DEMO consists of two programs - SPI DEMOM.SRC and SPI DEMOS.SRC, which correspond to the SPI Master and SPI Slave sides accordingly. These demo files have within them the two VPs described above SPI Master and SPI Slave.

There are 4 signal lines that connect the Master to the Slave:

Master-Out-Slave-In	MOSI signal line
Master-In-Slave-Out	MISO signal line
Serial Clock	SCLK signal line
Slave Select	$\overline{SS}$ signal line

The Slave Select ($\overline{SS}$) Line from Master to Slave signals the latter to start sending/receiving data. The word length is set to 16 bits in this demo (VP permits 1 to 16 bits). SPI virtual peripherals perform shifting in and out at the same time, as most hardware SPI implementations do.

The demo works as follows:

- Assuming that Slave has been initialized prior to Master, the Master sends the predefined word 256 times.The predefined words are in registers SPI_DEMO_DATA_MSB and SPIM_DEMO_DATA_LSB.
- The Slave SX is configured to perform the loop-back function, that is, every time Slave sends to the Master whatever it has received from the Master during the previous transmission.
- After the initial 256 transmissions Master switches into the loop-back mode.
- The data transfer of the predefined word can be easily observed with the help of an oscilloscope.

SPI clock period for this demo is 900 nsec - maximum clock rate supported by the SPIS (SPI Slave VP). Data transfer from the Master side is triggered by RTCC interrupt with a period of 25.6 microseconds.

To run the SPI demo program connect two SX28 microcontrollers in a master slave configuration as shown in figure 6-1, do not forget to connect a common ground. By connecting an oscilloscope to the signal lines it is possible to see the data being sent back and forth between the master and slave devices on each clock pulse.

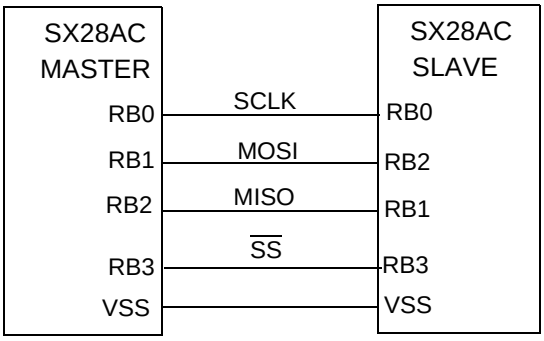

Figure 6-1. SPI Master-Slave Demo Program Configuration

Once you are familiar with the SPI demo, the master and slave VPs can be configured for your exact needs by writing to the appropriate registers. In Figure 6-2 a scope capture of the $\overline{SS}$ and SCLK was taken to show how the $\overline{SS}$ line is used to synchronize the clock and data.

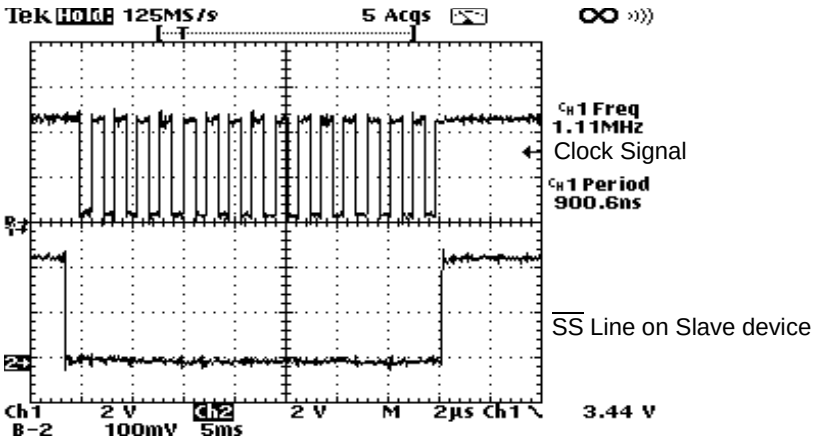

Figure 6-2. Clock and $\overline{SS}$ line on slave device

6.1 INTERFACE CONSIDERATIONS

To protect the integrity of data exchange when using synchronous serial communication, two aspects must be considered:

- Serial Data Exchange Timing
- Fan-out/fan-in requirements

Theoretically, infinite devices can access the same interface and be uniquely enabled sequentially in time. In practice, however the actual number of devices that can access the same serial interface depends on the following: System data transfer rate, system supply requirements, capacitive loading, and the fan-in requirements of the logic families or discrete devices to be interfaced.

