SPI Bus Interface

The SPI (Serial Peripheral Interface) bus interface is a very useful method of communicating at very high speeds between a microcontroller and other chips such as an external DA converter, an external AD converter, external memory, external counters for encoders or other microcontrollers. In the SPI bus protocol there is a Master that controls the data flow and a Slave that responds to the data flow. Most of the time the microcontroller is the Master and the external device is the Slave.

There are four wires that are necessary for SPI bus communication:

SCK — Clock signal that controls the data flow. This is a square wave signal that transfers and reads one bit of data per clock cycle.

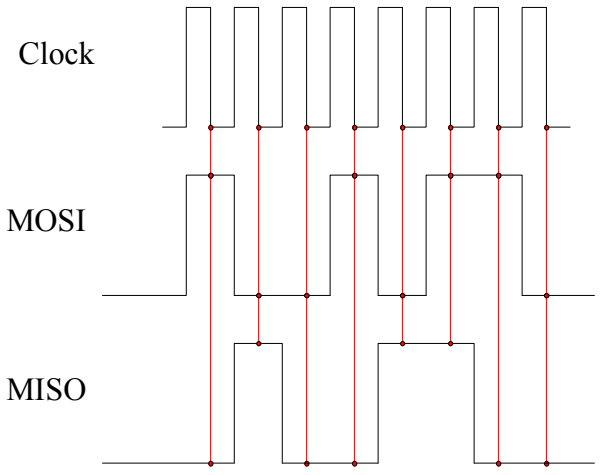
MOSI — The Master Out and Slave In (MOSI) signal transfers data from the Master to the Slave. During each clock cycle the MOSI pin will be either low or high indicating the transfer of a zero or a one from the Master to the Slave.

MISO — The Master In and Slave Out (MISO) signal transfers data from the Slave to the Master. During each clock cycle the MISO pin will be either low or high indicating the transfer of a zero or a one from the Slave to the Master.

The SPI bus contains one Master and can have many Slave devices connected to the SPI bus. So there needs to be a way for the Master to select a single Slave device to communicate with. The chip select pin (CS) is used to define which single slave device is chosen for communication.

CS — Chip Select (CS) pin is used to define which single slave device is chosen for communication. Each device has a separate CS line.

The SPI bus has four different modes that define the details of the communication between the Master and the Slave. The four different modes define the polarity of the clock signal and the time during the cycle when the data is read and when it is changed. For example the data could be changed on the rising edge of the clock pulse and read on the falling edge of the clock pulse as shown in the figure below. The figure shows 0b10010110 written to MOSI and 0b01001100 read on MISO.



The different Slave devices may have different SPI protocols. This is not a problem however because the SPI bus can be easily and quickly switched between modes.

The ATmega88 manual discusses the SPI bus in detail. (http://www.atmel.com/Images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet.pdf)

Communication between the SPI bus is accomplished through the following registers:

SPSR Sets various parameters of the SPI bus (see Appendix A)

SPCR Sets various parameters of the SPI bus (see Appendix A)

SPDR SPI data transfer register. This register is used to both transfer data to the Slave and return data from the Slave. Writing data to this register initiates the data transfer. At each clock pulse data is written to MOSI from the corresponding bit of SPDR and simultaneously read from MISO and placed into the corresponding bit of SPDR. So at the end of the data transfer the original contents of SPDR has been written to MOSI and the data transmitted to MISO is written to SPDR. See Figures 59 and 60 in Appendix A for additional insight into the SPI data transfer.

The following example illustrates setting up and writing to the SPI bus.

Programmer's Notepad - SPI_test_WIN_Avr_ATmega8.c

```
// SPI_test_WIN_Avr_ATmega8.c

#include <avr/io.h>
#define sbi(var, mask)   ((var) |= (uint8_t)(1 << mask))
#define cbi(var, mask)   ((var) &= (uint8_t)~(1 << mask))
#define SPIF 7

// SPI write read function
unsigned char spi_write_read(unsigned char spi_data)
{
SPDR=spi_data;
while ((SPSR & (1<<SPIF))==0);  // Wait until the data transfer is complete
return SPDR;
}

int main (void)
{
    unsigned char   spi_data_0;
    unsigned char   spi_data_1;
    unsigned char   dummy_read;

    DDRB=0b00101100;    //Set Output Ports for the SPI Interface
    DDRD=0b10000000;    //Set Output Ports for the Chip select

    // SPI initialization
    SPCR=0b01010010;
    SPSR=0b00000000;

    while(1)
    {
        spi_data_0 = 0b10001000;  //Set up first byte to write
        spi_data_1 = 0b00100101;  //Set up second byte to write

        cbi(PORTD,7);                                   // Activate the chip - set chip select to zero
        dummy_read = spi_write_read(spi_data_0);    // Write/Read first byte
        dummy_read = spi_write_read(spi_data_1);    // Write/Read second byte
        sbi(PORTD,7);                                   // Release the chip  - set chip select to one
    }

}
```

The first few lines of code in red include the standard AVR files and define the bit set and clear functions from the tutorial. The SPI write and read function is defined next. The while statement in this function waits until the data transfer is completed. At that point the data in spi_data has been transferred to MOSI and the data (if any) from MISI is returned from the function. The main function starts by declaring the necessary variables and then sets the necessary output ports using registers DDRB and DDRD. The SPI bus is then initialized using registers SPCR and SPSR. The following while loop

executes indefinitely. The data to be written to the SPI bus is defined (spi_data_0 and spi_data_1) . Next the chip select line is set to zero (cbi(PORTD,7)) to simulate a chip select, the two bytes of data are written and the chip select is set to one (sbi(PORTD,7)) to release the device.

Setup a version of this program and verify the proper operation of the SPI data bus by viewing the various signals (MOSI, SCK, CS) on the oscilloscope. Note that the microcontroller does not have to be hooked up to a device to test the SPI bus. Experiment with changing the various parameters of the SPI bus (clock speed, SPI mode, etc.) and verify the results.

Once the SPI bus is operating properly the next step is to interface a DA converter to the SPI bus. The DA converter we will be using is the Microchip MCP4922. (http://ww1.microchip.com/downloads/en/DeviceDoc/21897B.pdf).

Start by thoroughly reading the data sheet to understand how the data is transferred from the microcontroller to the DA and how to set the parameters on the DA chip. Once this is completed carefully build the circuit to interface the DA to your microcontroller. Start testing the DA by writing a single voltage value to the DA and reading the voltage on the oscilloscope. Next create a linearly increasing voltage value that cycles between 0 and 4095 and then is reset to zero. Verify this voltage on the oscilloscope.

Finally incorporate the previously developed AD conversion into the program and test the AD and DA by using the function generator to input a sine wave to the AD and then output the signal to the DA. Investigate different frequencies and wave types. What is the fastest cycle time (AD and DA) that you can achieve?

Appendix A: SPI Register Information

(http://www.atmel.com/dyn/resources/prod_documents/doc2486.pdf)

SPI Control Register – SPCR

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIE: SPI Interrupt Enable**

This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR Register is set and the if the global interrupt enable bit in SREG is set.

- **Bit 6 – SPE: SPI Enable**

When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

- **Bit 5 – DORD: Data Order**

When the DORD bit is written to one, the LSB of the data word is transmitted first.

When the DORD bit is written to zero, the MSB of the data word is transmitted first.

- **Bit 4 – MSTR: Master/Slave Select**

This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If $\overline{SS}$ is configured as an input and is driven low while MSTR is set, MSTR will be cleared,

and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI Master mode.

- **Bit 3 – CPOL: Clock Polarity**

When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle. Refer to Figure 59 and Figure 60 for an example. The CPOL functionality is summarized below:

Table 48. CPOL Functionality

CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

- **Bit 2 – CPHA: Clock Phase**

The settings of the clock phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK. Refer to Figure 59 and Figure 60 for an example. The CPHA functionality is summarized below:

Table 49. CPHA Functionality

CPHA	Leading Edge	Trailing Edge
0	Sample	Setup
1	Setup	Sample

- **Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0**

These two bits control the SCK rate of the device configured as a Master. SPR1 and SPR0 have no effect on the Slave. The relationship between SCK and the Oscillator Clock frequency f_{osc} is shown in the following table:

Table 50. Relationship Between SCK and the Oscillator Frequency

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

SPI Status Register – SPSR

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIF: SPI Interrupt Flag**

When a serial transfer is complete, the SPIF Flag is set. An interrupt is generated if SPIE in SPCR is set and global interrupts are enabled. If $\overline{SS}$ is an input and is driven low when the SPI is in Master mode, this will also set the SPIF Flag. SPIF is cleared by hardware when executing the corresponding interrupt Handling Vector. Alternatively, the SPIF bit is cleared by first reading the SPI Status Register with SPIF set, then accessing the SPI Data Register (SPDR).

- **Bit 6 – WCOL: Write COLlision Flag**

The WCOL bit is set if the SPI Data Register (SPDR) is written during a data transfer. The WCOL bit (and the SPIF bit) are cleared by first reading the SPI Status Register with WCOL set, and then accessing the SPI Data Register.

- **Bit 5..1 – Res: Reserved Bits**

These bits are reserved bits in the ATmega8 and will always read as zero.

- **Bit 0 – SPI2X: Double SPI Speed Bit**

When this bit is written logic one the SPI speed (SCK Frequency) will be doubled when the SPI is in Master mode (see Table 50). This means that the minimum SCK period will be 2 CPU clock periods. When the SPI is configured as Slave, the SPI is only guaranteed to work at $f_{osc}/4$ or lower.

The SPI interface on the ATmega8 is also used for Program memory and EEPROM downloading or uploading. See page 237 for Serial Programming and verification.

SPI Data Register – SPDR

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

The SPI Data Register is a Read/Write Register used for data transfer between the Register File and the SPI Shift Register. Writing to the register initiates data transmission. Reading the register causes the Shift Register Receive buffer to be read.

Data Modes

There are four combinations of SCK phase and polarity with respect to serial data, which are determined by control bits CPHA and CPOL. The SPI data transfer formats are shown in Figure 59 and Figure 60. Data bits are shifted out and latched in on opposite edges of the SCK signal, ensuring sufficient time for data signals to stabilize. This is clearly seen by summarizing Table 48 and Table 49, as done below:

Table 51. CPOL and CPHA Functionality

	Leading Edge	Trailing Edge	SPI Mode
CPOL = 0, CPHA = 0	Sample (Rising)	Setup (Falling)	0
CPOL = 0, CPHA = 1	Setup (Rising)	Sample (Falling)	1
CPOL = 1, CPHA = 0	Sample (Falling)	Setup (Rising)	2
CPOL = 1, CPHA = 1	Setup (Falling)	Sample (Rising)	3

Figure 59. SPI Transfer Format with CPHA = 0

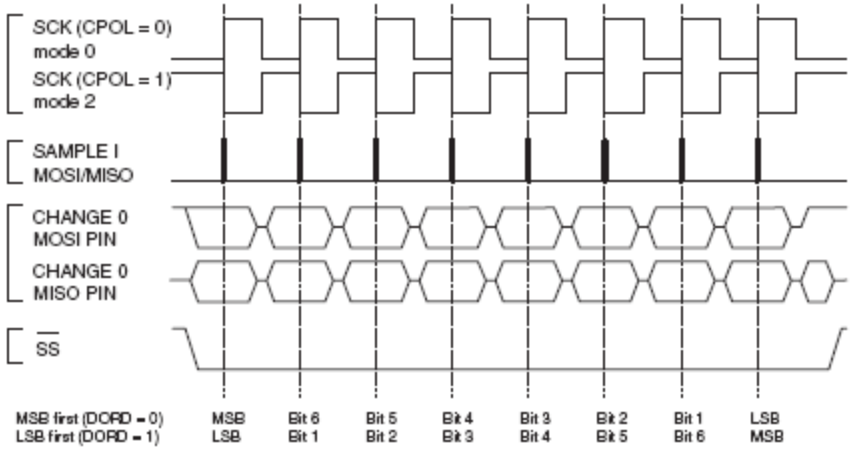
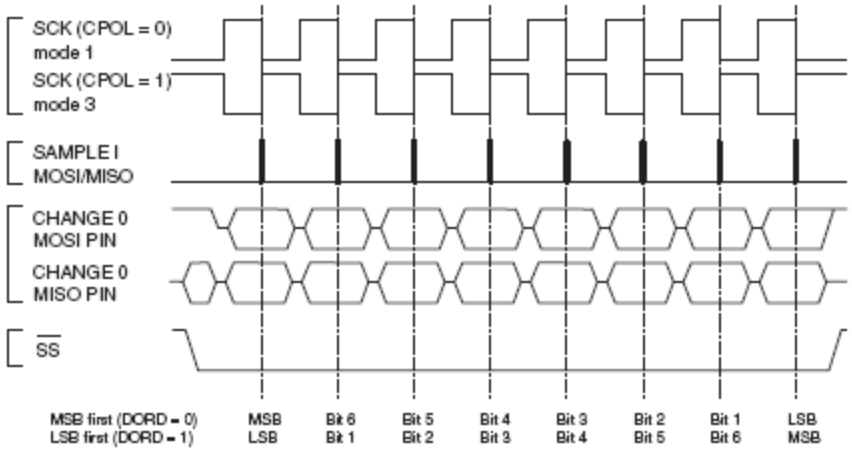


Figure 60. SPI Transfer Format with CPHA = 1



Appendix B: SPI bus Links

http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
http://www.embedded.com/story/OEG20020124S0116