Lab 12: Interfacing Devices using SPI

Objectives:

- To learn about Serial Peripheral Interfacing (SPI) and SPI programming
- Implementation of SPI using microcontroller (ATmega328P) in proteus.

Tools:

Software Tools:

- Atmel Studio/ AVR Studio
- Proteus ISIS
- AVR DUDESS

Hardware Tools:

Name	Value	Quantity
Arduino Nano	-	1
Breadboard	-	1
MCP4821	-	1

Table 12.1: List of Components

PRE-LAB

Introduction to Serial Peripheral Interfacing (SPI)

Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between microcontrollers and small peripherals such as shift registers, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device you wish to talk to.

A common serial port, the kind with TX and RX lines, is called "asynchronous" because there is no control over when data is sent or any guarantee that both sides are running at precisely the same rate. To work around this problem, asynchronous serial connections add extra start and stop bits to each byte help the receiver sync up to data as it arrives. Both sides must also agree on the transmission speed (such as 9600 bits per second) in advance

SPI works in a slightly different manner. It's a "synchronous" data bus, which means that it uses separate lines for data and a "clock" that keeps both sides in perfect sync. The clock is an oscillating signal that tells the receiver exactly when to sample the bits on the data line. This could be the

rising (low to high) or falling (high to low) edge of the clock signal; the datasheet will specify which one to use. When the receiver detects that edge, it will immediately look at the data line to read the next bit.

HOW SPI WORKS:

SPI consists of two shift or more registers. One at master side, others at slave side.In SPI, only one side generates the clock signal .The side that generates the clock is called the "master", and the other side is called the "slave". There is always only one master (which is almost always your microcontroller), but there can be multiple slaves.When data is sent from the master to a slave, it's sent on a data line called MOSI, for "Master Out / Slave In". If the slave needs to send a response back to the master, the master will continue to generate a prearranged number of clock cycles, and the slave will put the data onto a third data line called MISO, for "Master In / Slave Out".

Slave Select (SS)

Slave select line tells the slave that it should wake up and receive / send data and is also used when multiple slaves are present to select the one you'd like to talk to.

The SS line is normally held high, which disconnects the slave from the SPI bus just before data is sent to the slave, the line is brought low, which activates the slave. When you're done using the slave, the line is made high again.

SPI Read and Write:

In connecting a device with SPI bus to a microcontroller, we use microcontroller as master and SPI device as slave. Microcontroller generates SCLK which is SCLK pin of SPI. The information address and data) is transferred between microcontroller and SPI device in group of 8 bits, where address byte is followed immediately by data byte. To distinguish between the read and write operations, the D7 bit of address byte is always 1 for write, while for read, D7 bit is low.

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
$\overline{SS}$	User Defined	Input

Figure 12.1 SPI Pins

In Lab:

SPI Programming in AVR:

In AVR, three Registers are associated with SPI. They are SPSR (SPI Status Register), SPCR (SPI Control Register) and SPDR (SPI Data Register).

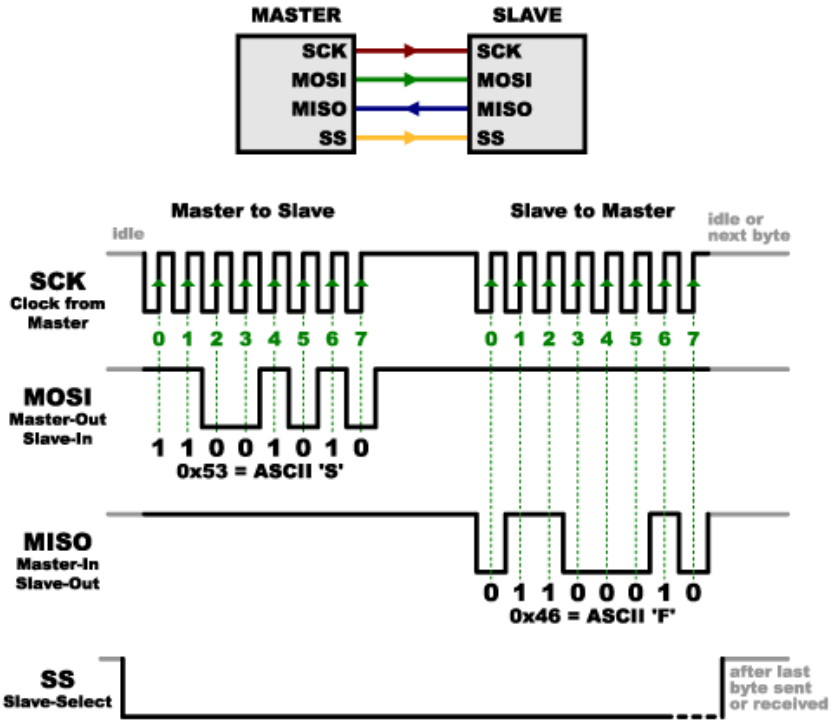

Figure 12.2: Working of SPI

SPI Control Register 0:

SPI Control Register 0 is shown below.

Name: SPCR0
Offset: 0x4C
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x2C

Bit	7	6	5	4	3	2	1	0
	SPIE0	SPE0	DORD0	MSTR0	CPOL0	CPHA0	SPR01	SPR00
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SPI Status Register 0:

SPI Status Register 0 is shown below:

Name: SPSR0
Offset: 0x4D
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x2D

Bit	7	6	5	4	3	2	1	0
	SPIF0	WCOL0						SPI2X0
Access	R	R						R/W
Reset	0	0						0

SPI Data Register 0:

The SPI Data register is a Read/write register. It contains data to be transmitted or received data. Writing to SPDR register initiates data transmission.

Name: SPDR0
Offset: 0x4E
Reset: 0xXX
Property: When addressing as I/O Register: address offset is 0x2E

Bit	7	6	5	4	3	2	1	0
	SPID[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Clock Polarity and phase in SPI:

In SPI communication, the master and slaves must agree on clock polarity and phase. Table 12.3 shows options of clock polarity and phase.

CPOL	CPHA	Data Read and Change Time	SPI Mode
0	0	Read on rising edge, changed on a falling edge	0
0	1	Read on falling edge, changed on a rising edge	1
1	0	Read on falling edge, changed on a rising edge	2
1	1	Read on rising edge, changed on a falling edge	3

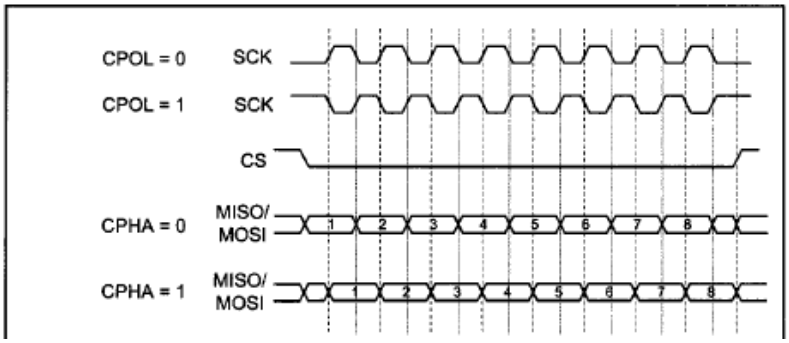

Table 12.2: SPI clock polarity and phase

MCP4821 DAC:

MCP4821 is a single channel 12-bit Digital-to-Analog converter (DAC) with internal voltage reference. This device offers high accuracy and low power consumption, and is available in various packages. Communication with the device is accomplished via a simple serial interface using SPI protocols.

Some of its features are:

- 12-bit Resolution
- Single Channel Voltage Output
- 2.7V to 5.5V Operation
- Operating Current 330 µA (typ)
- Internal Voltage Reference 2.048V
- Selectable Unity or 2x Gain Output
- works with SPI mode 0

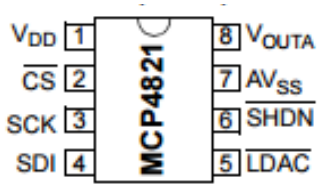

Figure12.6: Pin diagram of MCP4821

$\overline{SHDN} = V_{DD}$ for normal operation

$$V_{out} = \frac{2.048 \times D_n}{2^{12}} \times G$$ (where G is gain)

MCP4821 DAC SPI Register

Some features of MCP4821 DAC SPI Register are:

- Only one 16-bit data register
  a) Bit 15 should always be written with zero
  b) Gain G = 2 if GA = and Gain G = 1 if GA = 1
  c) $\overline{SHDN}$ = 1 to activate device, otherwise shutdown

d) D11 down to D0 the digital input

W-x	W-x	W-x	W-0	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x
0	—	$\overline{GA}$	$\overline{SHDN}$	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
bit 15															bit 0

Fig 12.7 16 bit data register

- MSB is shifted in first then LSB is shifted

- AVR has SPI module with 8-bit shift register
  a) So AVR need to send 16-bit data in two portions
  b) First send upper byte, then lower byte

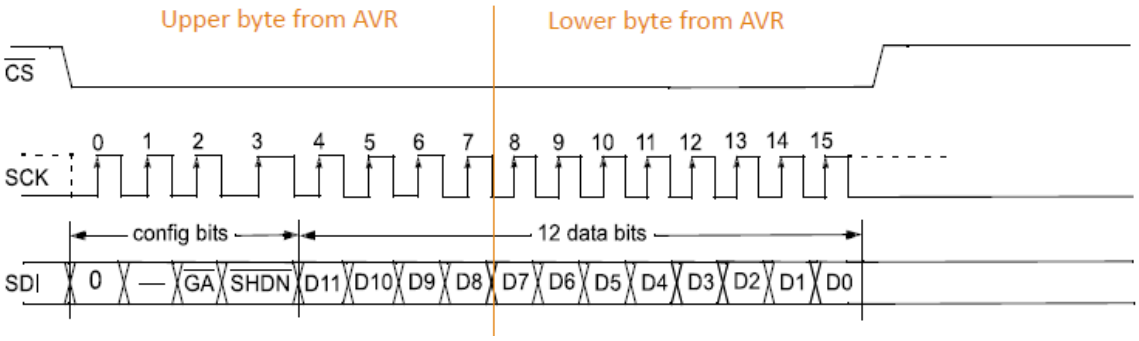

Fig 12.8: Sending data from AVR to MCP 4821 Data register

In Lab Task:

The given code contains functions to write bytes to SPI and writing to DAC. Your task is to write code to send a sequence of numbers to the DAC to generate a saw-tooth wave. Use gain =2x

Code:

Complete and build the given code.

```
#include <avr\io.h>                // Most basic include files
#include <avr\interrupt.h>         // Add the necessary ones
#include <avr\signal.h>            // here
#define F_CPU 16000000
#define MOSI 3
#define SCK 5
#define SS 2
void SPI_byte_send(char ebyte);
void write_SPI_DAC(unsigned int val);
void SPI_Init(void);
int main(void)
{
    unsigned int val = 0;
      DDRC=0xFF;
      SPI_Init();
    while (1)
    {

            // write code to generate a sawtooth waveform on DAC
    }
}

void SPI_byte_send(char ebyte)
{
```

```
      //This function send one byte through AVR SPI module

      //This function should ensure that existing communication is
complete

      //PORTC = SPSR;



      SPDR = ebyte;

      while(!(SPSR & (1<<SPIF)));

}



void write_SPI_DAC(unsigned int val)

{

      //clear SS

      PORTB &= ~(1<<SS);



      //Send upper byte (0b0000xxxx) where xxxx is upper four bits of
Digital data

      //Upper nible zero suggest GA'=0 SHDN'=0

      SPI_byte_send(0b00110000 | ( (val>>8) & 0x0F ));      //send
upper byte



      //Send lower 8 bits of digital data

      SPI_byte_send( val & 0xFF );  //send lower byte



      //Set SS

      PORTB |= (1<<SS);



      //_delay_us(50);



}



void SPI_Init()

{
```

```
      // Write code to initialize SPI
}
```

Simulation:

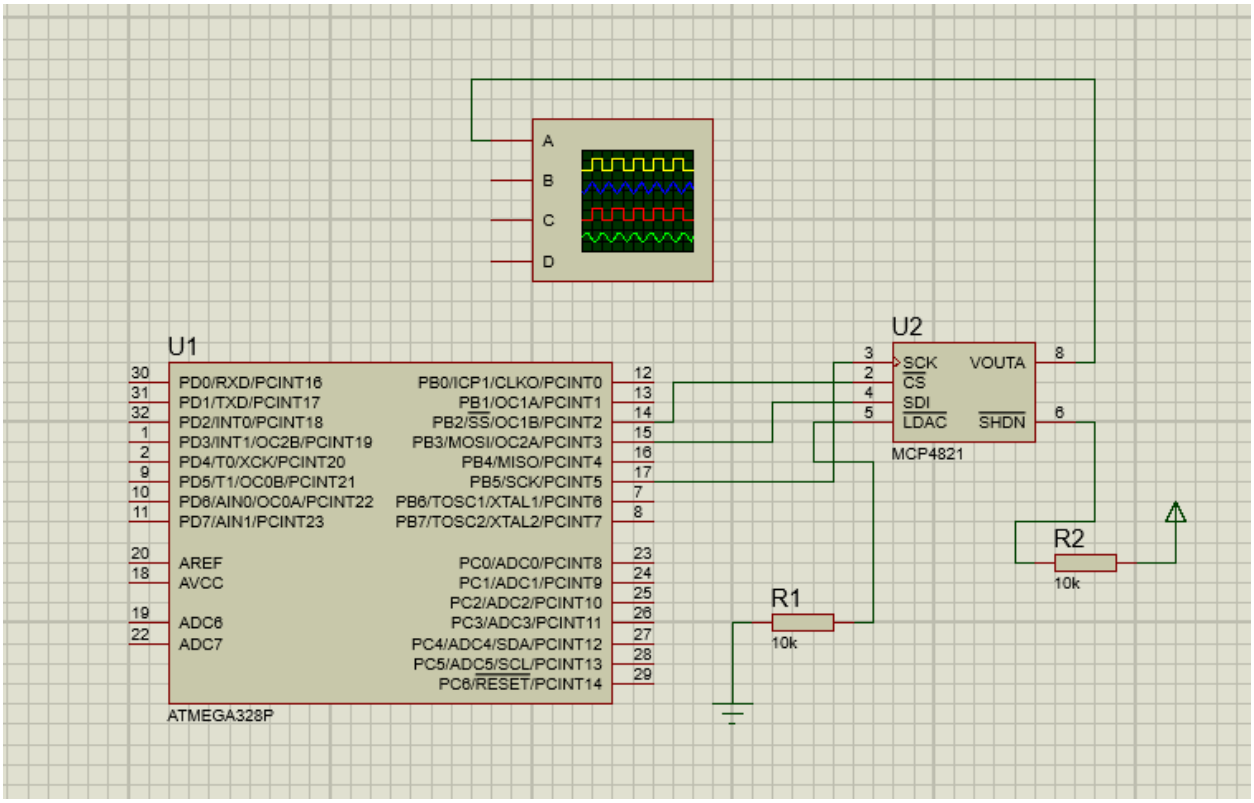

Figure 11.8: Schematic for In Lab task

Critical Analysis / Conclusion

(By Student about Learning from the Lab)

Lab Assessment		
Pre Lab	**/1**	**/10**
In Lab	**/5**	
Post Lab	**/4**	
Instructor Signature and Comments		