# Experiment-11

**Name**:- Aryan Langhanoja
**Enrollment No**:- 92200133030

**Aim:** Hands-on experimentation of SPI Programming with ATMEGA32 in C .

**Objectives:** After successfully completion of this experiment students will be able to,

- Use C language for ATMega32 microcontroller programming on AVRStudio.
- Experiment with SPI programming with ATMega32 on ATMega32 AVR Development Board.

**Equipment required:**

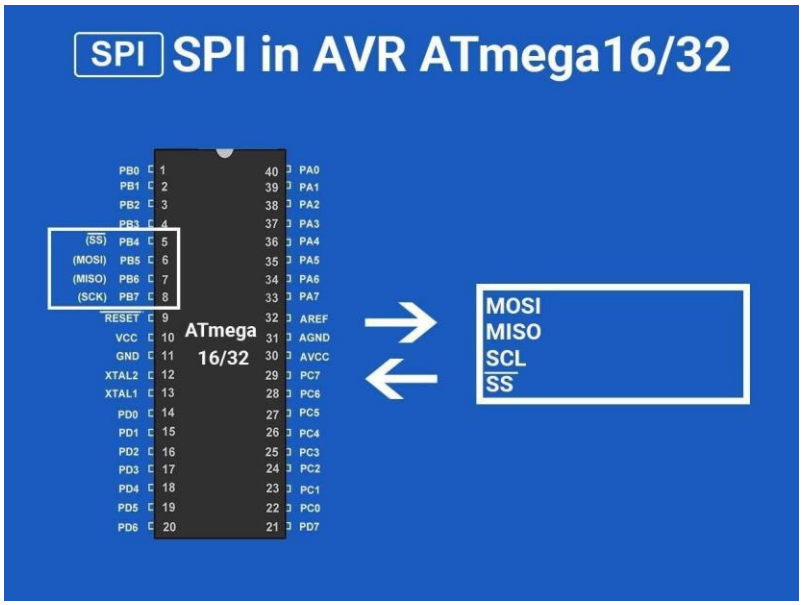
- Windows7 or later based host computer
- ATMega32 Development board
- USBasp Programmer
- Jumper Wires
- Peripherals

**Software required:**

- AVR Studio7 installation setup
- USBasp driver installation setup

**Theory:**

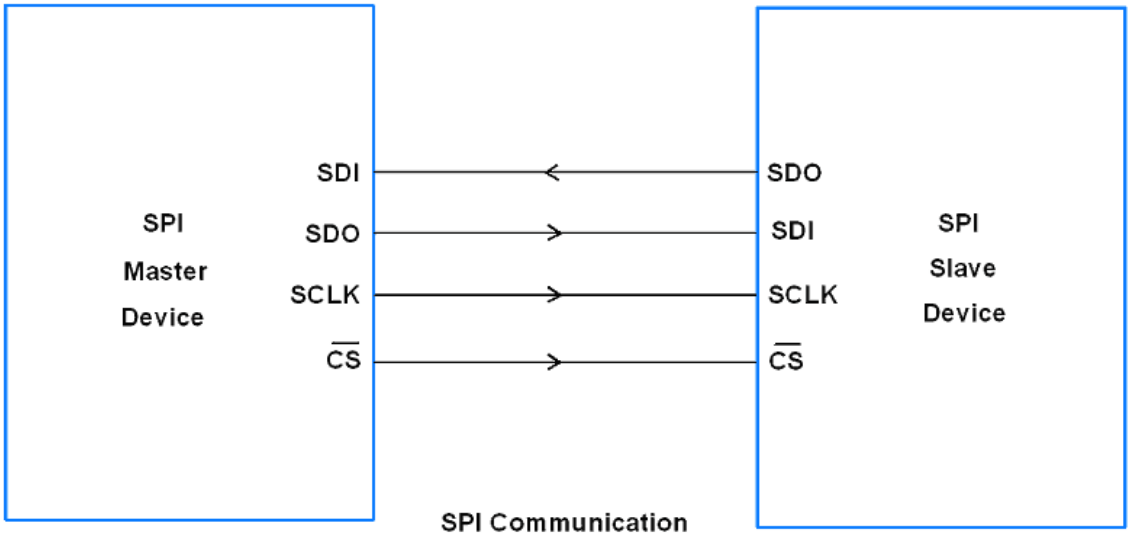
**Basics of SPI:**




The Serial Peripheral Interface (SPI) is a bus interface connection protocol originally started by Motorola Corp. It uses four pins for communication.

- SDI (Serial Data Input)
- SDO (Serial Data Output)
- SCLK (Serial Clock)
- CS (Chip Select)

It has two pins for data transfer called SDI (Serial Data Input) and SDO (Serial Data Output). SCLK (Serial Clock) pin is used to synchronize data transfer and Master provides this clock. CS (Chip Select) pin is used by the master to select the slave device.




SPI Communication

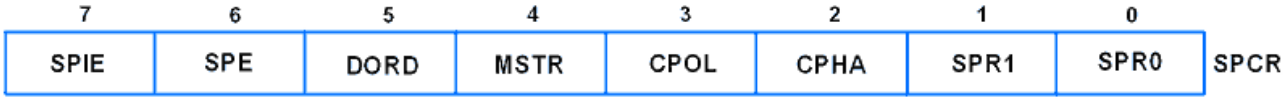
SPI devices have 8-bit shift registers to send and receive data. Whenever a master needs to send data, it places data on the shift register and generates a required clock. Whenever a master wants

to read data, the slave places the data on the shift register and the master generates a required clock. Note that SPI is a full-duplex communication protocol i.e. data on master and slave shift registers get interchanged at the same time.

AVR ATmega16 uses three registers to configure SPI communication that are SPI Control Register, SPI Status Register and SPI Data Register.

Let's see these Registers.

SPCR: SPI Control Register




Bit 7 – SPIE: SPI Interrupt Enable bit

1 = Enable SPI interrupt.

0 = Disable SPI interrupt.

Bit 6 – SPE: SPI Enable bit

1 = Enable SPI.

0 = Disable SPI.

Bit 5 – DORD: Data Order bit

1 = LSB transmitted first.

0 = MSB transmitted first.

Bit 4 – MSTR: Master/Slave Select bit

1 = Master mode.

0 = Slave Mode.

Bit 3 – CPOL: Clock Polarity Select bit

1 = Clock start from logical one.

0 = Clock start from logical zero.

Bit 2 – CPHA: Clock Phase Select bit

1 = Data sample on trailing clock edge.

0 = Data sample on the leading clock edge.

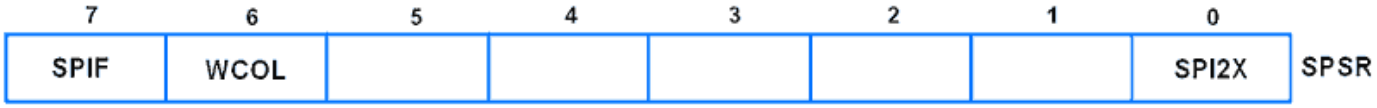
Bit 1:0 – SPR1: SPR0 SPI Clock Rate Select bits

The below table shows the SCK clock frequency select bit setting.

| SPI2X | SPR1 | SPR0 | SCK Frequency |
|---|---|---|---|
| 0 | 0 | 0 | Fosc/4 |
| 0 | 0 | 1 | Fosc/16 |
| 0 | 1 | 0 | Fosc/64 |

| SPI2X | SPR1 | SPR0 | SCK Frequency |
|---|---|---|---|
| 0 | 1 | 1 | Fosc/128 |
| 1 | 0 | 0 | Fosc/2 |
| 1 | 0 | 1 | Fosc/8 |
| 1 | 1 | 0 | Fosc/32 |
| 1 | 1 | 1 | Fosc/64 |

SPSR: SPI Status Register




Bit 7 – SPIF: SPI interrupt flag bit

- This flag gets set when the serial transfer is complete.
- Also gets set when the SS pin is driven low in master mode.
- It can generate an interrupt when SPIE bit in SPCR and a global interrupt is enabled.

Bit 6 – WCOL: Write Collision Flag bit

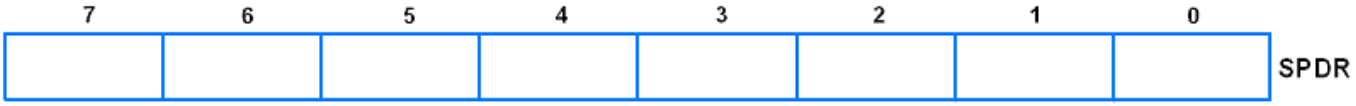
- This bit gets set when SPI data register writes occurs during previous data transfer.

Bit 5:1 – Reserved Bits

Bit 0 – SPI2X: Double SPI Speed bit

- When set, SPI speed (SCK Frequency) gets doubled.

SPDR: SPI Data Register




- SPI Data register used to transfer data between the Register file and SPI Shift Register.
- Writing to the SPDR initiates data transmission.

When the device is in master mode

- Master writes data byte in SPDR. Writing to SPDR starts data transmission.
- 8-bit data starts shifting out towards slave and after the complete byte is shifted, SPI clock generator stops, and SPIF bit gets set.

When the device is in slave mode

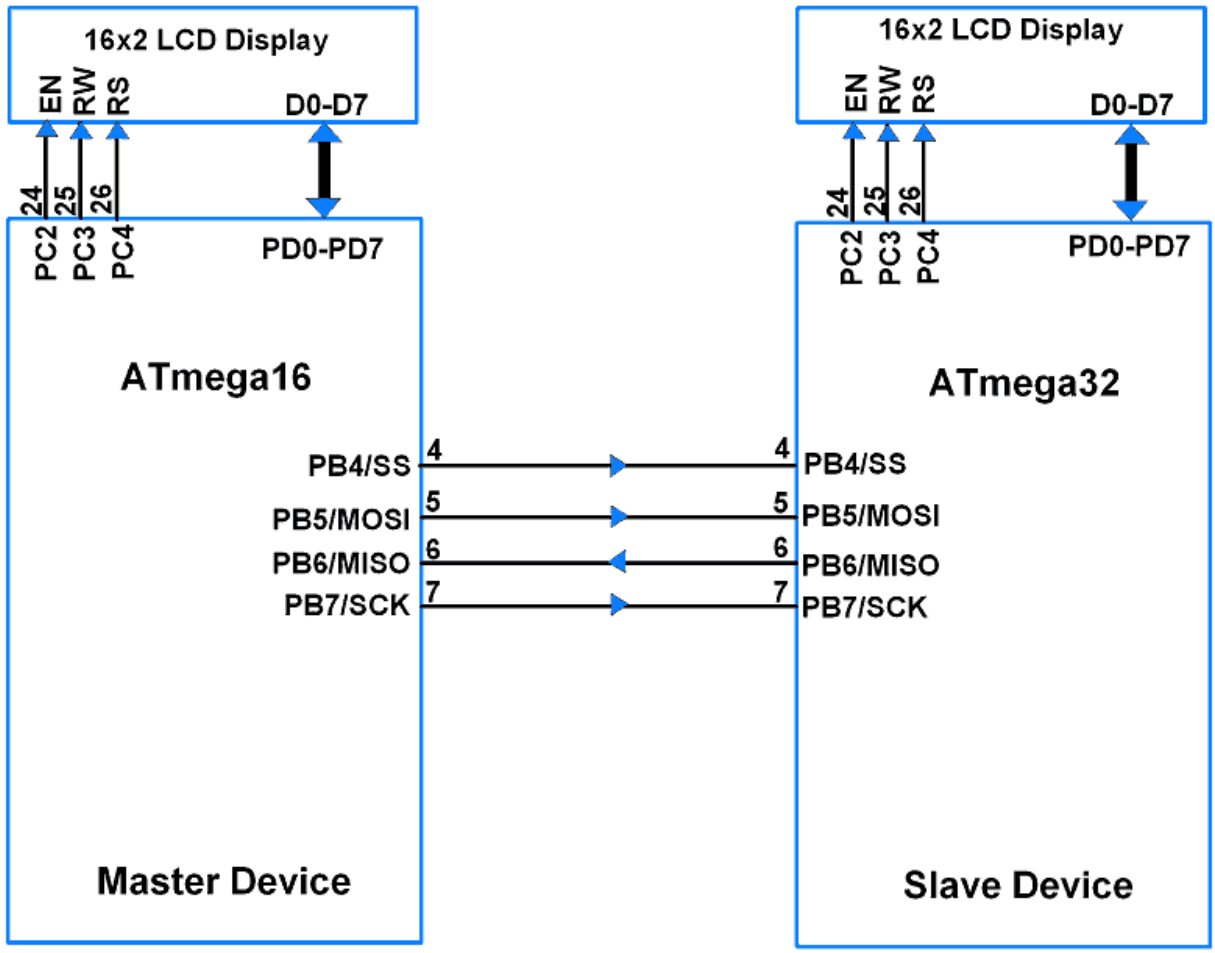
- THE Slave SPI interface remains in sleep as long as the SS pin is held high by the master.
- It activates only when the SS pin is driven low. Data is shifted out with incoming SCK clock from master during a write operation.
- SPIF is set after the complete shifting of a byte.

SS pin functionality Master mode

- In master mode, the SS pin is used as a GPIO pin.
- Make the SS pin direction as output to select a slave device using this pin.
- Note that if the SS pin configured as input then it must be set high for master operation.
- If it is set as input in master mode and gets driven low by an external circuit, then the SPI system recognizes this as another master selecting SPI as a slave due to its active low behavior.
- This will clear the MSTR bit in the SPCR register and SPI turns in slave mode.

SS pin functionality Slave mode

- In slave mode, the SS pin is always configured as an input.
- When it is low SPI activates.
- And when it is driven high SPI logic gets reset and does not receive any incoming data.




**SPI Master-Slave Communication**

SPI Code for Master Device:

```
#define F_CPU 8000000UL                    /* Define CPU Frequency 8MHz */
#include <avr/io.h>                        /* Include AVR std. library file */
#include <util/delay.h>                    /* Include Delay header file */
#include <stdio.h>                  /* Include Std. i/p o/p file */
#include <string.h>                        /* Include String header file */
#include "LCD_16x2_H_file.h"               /* Include LCD header file */
#include "SPI_Master_H_file.h"             /* Include SPI master header file */

int main(void)
```

```
{
        uint8_t count;
        char buffer[5];

        LCD_Init();
        SPI_Init();

        LCD_String_xy(1, 0, "Master Device");
        LCD_String_xy(2, 0, "Sending Data:    ");
        SS_Enable;
        count = 0;
        while (1)                       /* Send Continuous count */
        {
                SPI_Write(count);
                sprintf(buffer, "%d   ", count);
                LCD_String_xy(2, 13, buffer);
                count++;
                _delay_ms(500);
        }

}
```

Program for SPI Slave Device:

```
#define F_CPU 8000000UL                         /* Define CPU Frequency 8MHz */
#include <avr/io.h>                     /* Include AVR std. library file */
#include <util/delay.h>                 /* Include Delay header file */
#include <stdio.h>                  /* Include std. i/p o/p file */
#include <string.h>                     /* Include string header file */
#include "LCD_16x2_H_file.h"            /* Include LCD header file */
#include "SPI_Slave_H_file.h"           /* Include SPI slave header file */

int main(void)
{
        uint8_t count;
        char buffer[5];
```

```
        LCD_Init();
        SPI_Init();

        LCD_String_xy(1, 0, "Slave Device");
        LCD_String_xy(2, 0, "Receive Data:    ");
        while (1)                       /* Receive count continuous */
        {
                count = SPI_Receive();
                sprintf(buffer, "%d   ", count);
                LCD_String_xy(2, 13, buffer);
        }

}
```

Circuit :-