AVR Interfacing

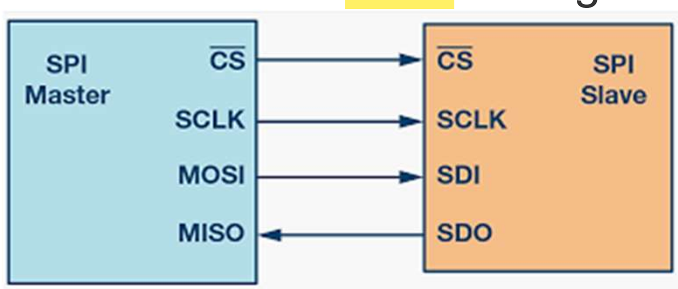
Serial Peripheral Interface (SPI)

Agenda

- Introduction
- Registers Description
- Programming

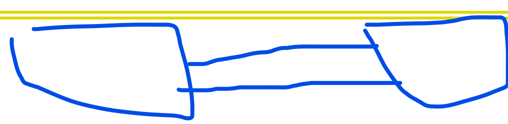
Introduction

- The Serial Peripheral Interface (SPI) is a bus interface connection protocol originally started by Motorola Corp. It uses four pins for communication.
  - **MISO (Master In Slave Out)**: Master receives data and slave transmits data through this pin.
  - **MOSI (Master Out Slave In):** Master transmits data and slave receives data through this pin.
  - **SCK (Shift Clock):** Master generates this clock for the communication, which is used by slave. Only master can initiate serial clock.
  - **SS (Slave Select):** Master can select slave through this pin.

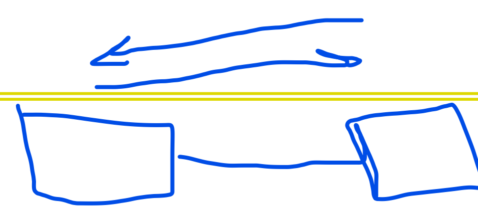


- SPI devices have 8-bit shift registers to send and receive data.
  - Whenever master need to send data, it places data on shift register and generate required clock.
  - Whenever master want to read data, slave places data on shift register and master generate required clock.

Note that SPI is full duplex communication protocol i.e. data on master and slave shift registers get interchanged at a same time.



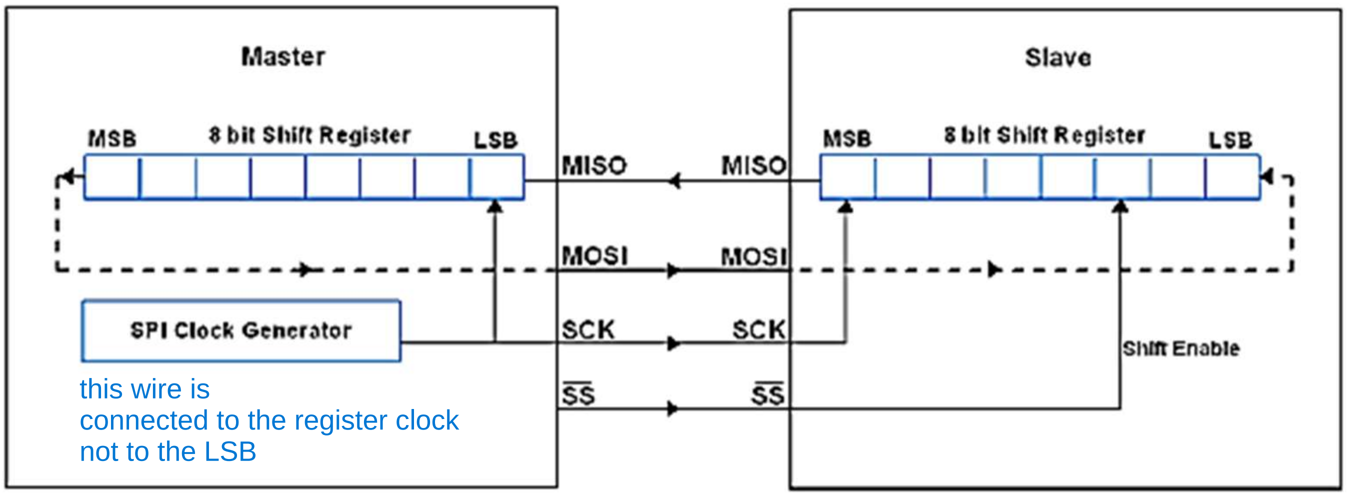
full duplex



half duplex

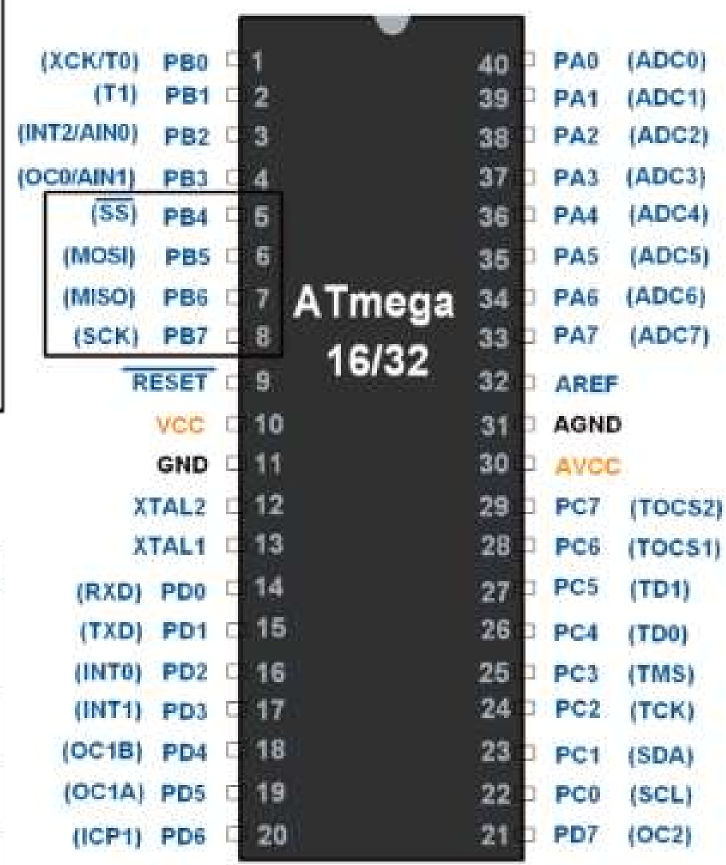
Introduction

The interconnection between master and slave using SPI is shown in below figure.



SPI Master Slave Interconnection

AVR ATmega16/32 SPI Pins



Pins Configurations:

SPI Pins	Pin No. on ATmega16 chip	Pin Direction in master mode	Pin Direction in slave mode
MISO	7	Input	Output
MOSI	6	Output	Input
SCK	8	Output	Input
SS	5	Output	Input

these number of pins not the same on the arduino

Registers Description

- AVR ATmega16/32 uses three registers to configure SPI communication that are SPI Control Register, SPI Status Register and SPI Data Register.
  - **SPCR: SPI Control Register**

7	6	5	4	3	2	1	0	
SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR

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 leading clock edge.

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

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
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

Introduction

if the clock was 0 then the leading edge will be 1, so it is the rising, and vice versa

CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

this determines when we are going to read and when we will write

CPHA	Leading Edge	Trailing Edge
0	Sample read	Setup
1	Setup shift	Sample

number of combinations?
correct -> 4.

Introduction

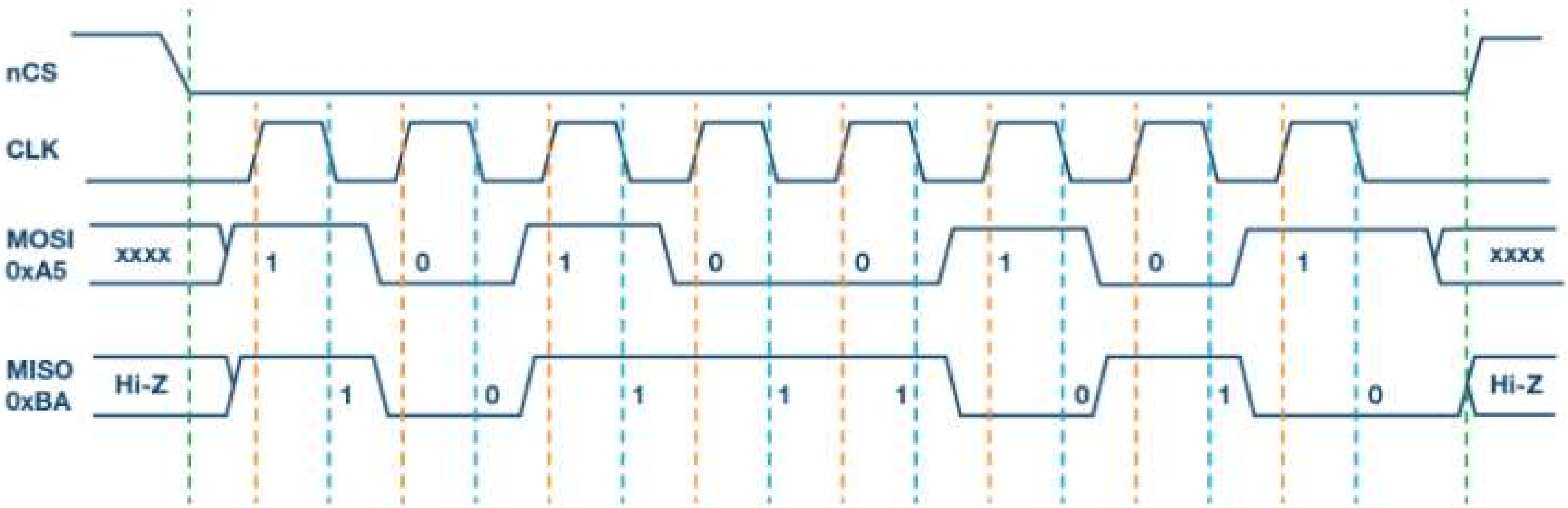


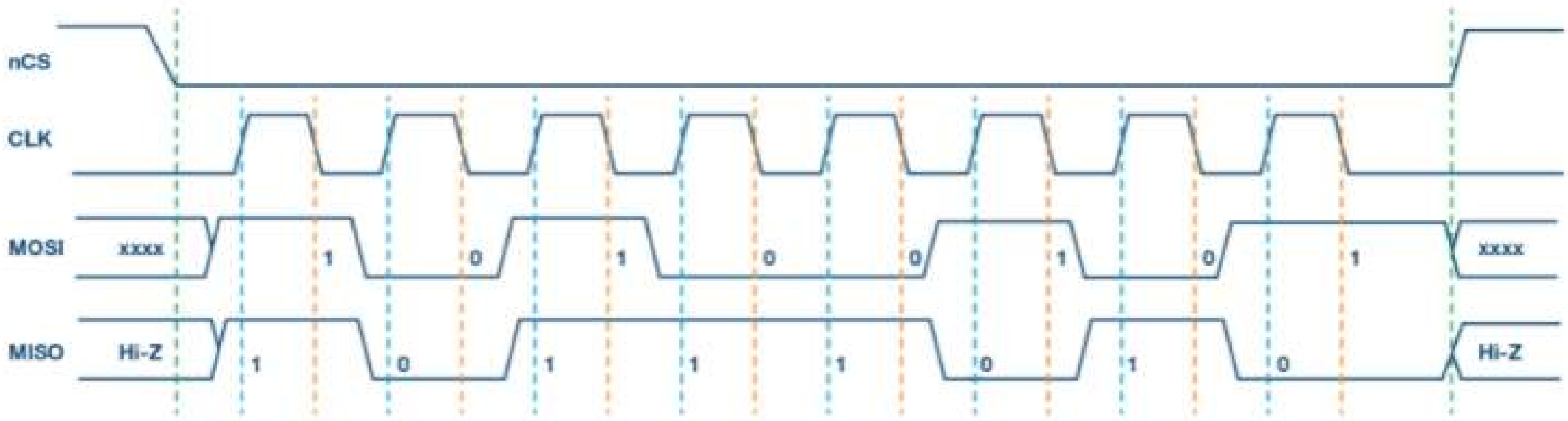
Figure 2. SPI Mode 0, CPOL = 0, CPHA = 0: CLK idle state = low, data sampled on rising edge and shifted on falling edge.

ehna olna, msh bn3ml generation ll clk gher whna bnb3t, tb lw mbnb3tsh el clock 7aletha hatb2a eh? hya de el idle state.

sampled-> orange
shifted -> blue

Introduction

byb3t el data m3 el blue, w by2ra mn el orange



sampled-> read

Figure 3. SPI Mode 1, CPOL = 0, CPHA = 1: CLK idle state = low, data sampled on the falling edge and shifted on the rising edge.

Introduction

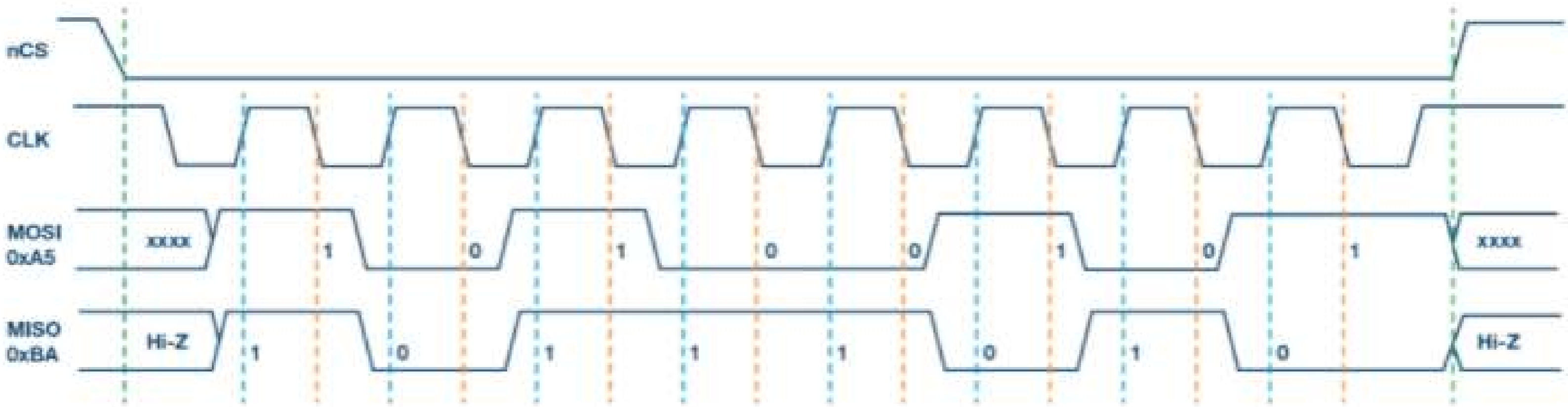


Figure 4. SPI Mode 3, CPOL = 1, CPHA = 1: CLK idle state = high, data sampled on the falling edge and shifted on the rising edge.

Introduction

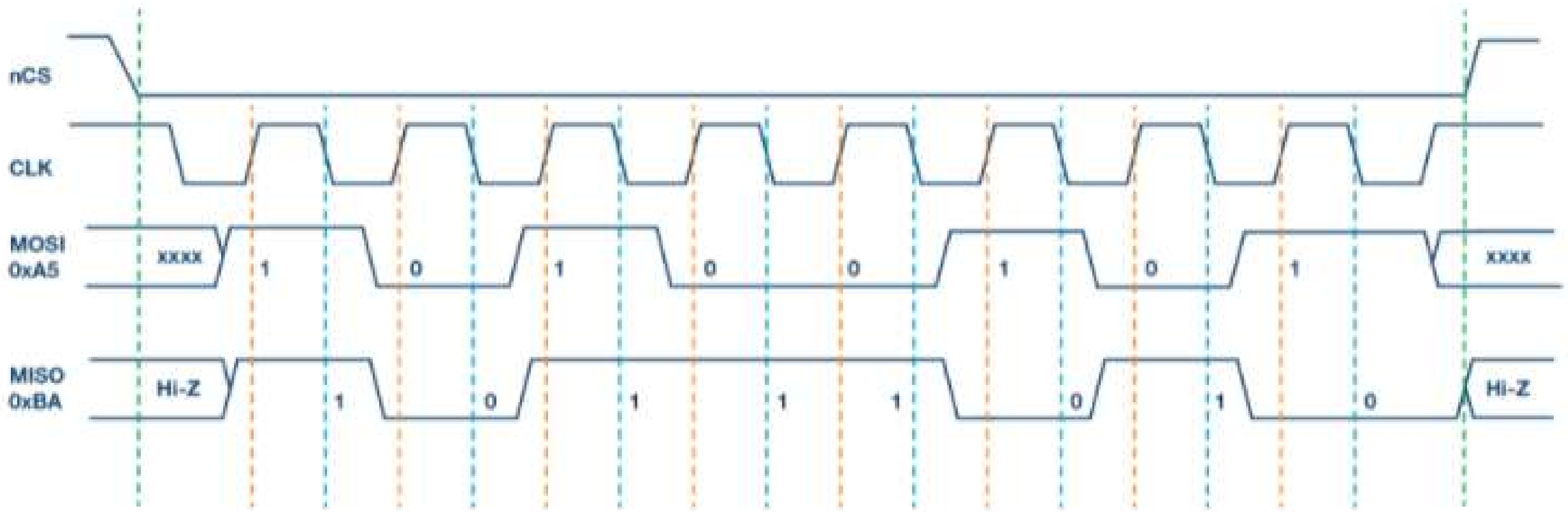


Figure 5. SPI Mode 2, CPOL = 1, CPHA = 0: CLK idle state = high, data sampled on the rising edge and shifted on the falling edge.

Registers Description

- AVR ATmega16/32 uses three registers to configure SPI communication that are SPI Control Register, SPI Status Register and SPI Data Register.
  - **SPSR: SPI Status Register**

7	6	5	4	3	2	1	0	
SPIF	WCOL						SPI2X	SPSR

Bit 7 – SPIF: SPI interrupt flag bit

This flag gets set when serial transfer is complete.

Also gets set when SS pin is driven low in master mode.

Bit 6 – WCOL: Write Collision Flag bit

This bit gets set when SPI data register (SPDR) is written during precious data transfer.

Bit 5:1 – Reserved Bits

Bit 0 – SPI2X: Double SPI Speed bit

When set, SPI speed (SCK frequency) get doubled.

Registers Description

- AVR ATmega16/32 uses three registers to configure SPI communication that are SPI Control Register, SPI Status Register and SPI Data Register.
  - ➢ **SPDR: SPI Data Register**

7	6	5	4	3	2	1	0	
								SPDR

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

- When device is in master mode
  - ➢ Master writes data byte in SPDR. Writing to SPDR starts the 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 device is in slave mode
  - ➢ Slave SPI interface remains in sleep as long as SS pin is held high by master.
  - ➢ It activates only when SS pin is driven low. Data is shifted out with incoming SCK clock from master during write operation.
  - ➢ SPIF is set after complete shifting of a byte.

Registers Description

- SS pin functionality Master mode
  - In master mode SS pin is used as GPIO pin.
  - Make SS pin direction as output to select slave device using this pin.
- SS pin functionality Slave mode
  - In slave mode SS pin is always configured as input.
  - When it is low SPI activates.
  - And when it driven high SPI logic gets reset and does not receive any incoming data.

ss -> slave select, de blnesba lel slave hya input, lw kemtha b2t 0 sa3tha el slave byshtghl
blnsba ll master hya output, lama byktb 3leha 0, hwa kda by5tar slave mo3ayn eno yeshghlo
lw 3ndy kaza slave sa3tha hyb2a 3ndy kaza pin, w b5tar benhom

Programming

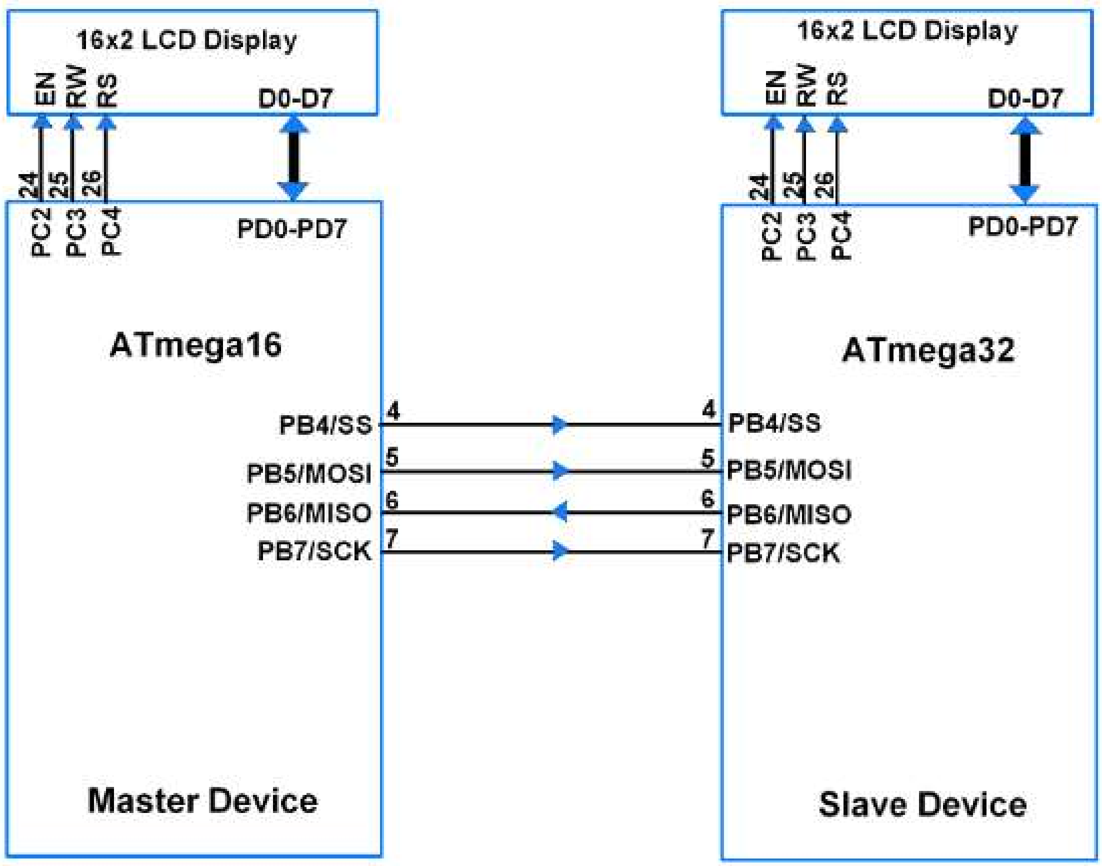
- Example:
- Let's do SPI communication using AVR family based ATmega16 (Master) and ATmega32 (Slave). Master will send continuous count to Slave. Display the sent and received data on 16x2 LCD.

lets explain it on arduino
this is better for labs purpose.

1. define the pins because they are not defined in the libraries

2. connect them to a common ground by connecting the 2 gnd of the 2 chips together.

3. MOSI (MASTER) -> MOSI(SLAVE) same for all other pins



lets skip this and explain the lab requirment better.

Programming

- Example:

  Let's do SPI communication using AVR family based ATmega16 (Master) and ATmega32 (Slave). Master will send continuous count to Slave. Display the sent and received data on 16x2 LCD.

  Let's first program Master device

  - SPI Master Initialization steps

    To initialize as Master, do the following steps

    - ✓ Make MOSI, SCK and SS pins directions as output.
    - ✓ Make MISO pin direction as input.
    - ✓ Make SS pin High.
    - ✓ Enable SPI in Master mode by setting SPE and MSTR bits in SPCR register.
    - ✓ Set SPI Clock Rate Bits combination to define SCK frequency.

```
void SPI_Init()                       /* SPI Initialize function */
{
    DDRB |= (1<<MOSI)|(1<<SCK)|(1<<SS); /* Make MOSI, SCK, SS
                                        as Output pin */
    DDRB &= ~(1<<MISO);               /* Make MISO pin
                                        as input pin */
    PORTB |= (1<<SS);                 /* Make high on SS pin */
    SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);    /* Enable SPI in master mode
                                        with Fosc/16 */
    SPSR &= ~(1<<SPI2X);                  /* Disable speed doubler */
}
```

Programming

- SPI Master Write steps
  - ✓ Copy data to be transmitted in SPDR register.
  - ✓ Wait until transmission is complete i.e. poll SPIF flag to become High.
  - ✓ While SPIF flag gets set read SPDR using flush buffer.
  - ✓ SPIF bit is cleared by H/W when executing corresponding ISR routine.
  - ✓ Note that to clear SPIF bit, need to read SPIF and SPDR registers alternately.

❖SPI_Write function

Input argument: It has input argument of data to be transmit.

```
void SPI_Write(char data)       /* SPI write data function */
{
    char flush_buffer;
    SPDR = data;                /* Write data to SPI data register */
    while(!(SPSR & (1<<SPIF))); /* Wait till transmission complete */
    flush_buffer = SPDR;        /* Flush received data */
    /* Note: SPIF flag is cleared by first reading SPSR (with SPIF set) and
    then accessing SPDR hence flush buffer used here to access SPDR after SPSR read */
}
```

34an a3ml clear ll flag, lazm a7ot el SPDR gowa el flush_buffer

aw te3ml ISR 34an t3mlo clear automatic

Programming

- ➢ SPI Master Read steps
  - ✓ Since writing to SPDR generates SCK for transmission, write dummy data in SPDR register.
  - ✓ Wait until transmission is completed i.e. poll SPIF flag till it becomes High.
  - ✓ While SPIF flag gets set, read requested received data in SPDR.
- ❖ SPI_Read function

  Return value: It returns received char data type.

```
char SPI_Read()                     /* SPI read data function */
{
    SPDR = 0xFF;
    while(!(SPSR & (1<<SPIF)));  /* Wait till reception complete */
    return(SPDR);                 /* Return received data */
}
```

dummy values

el transmission msh bybd2 8er lama a7ot 7aga fl SPDR, fa hena bema eny bast2bl, fa b7ot dummy value fl SPDR, 34an bs ast2bl el data feha, w bstna l7d ma el reception ye5ls w arg3 el SPDR fl akher

Programming

➢ Program for SPI 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);
    }
}
```

Programming

Now Program for Slave device:

➢ SPI Slave Initialization steps

- ✓ Make MOSI, SCK and SS pins direction of device as input.
- ✓ Make MISO pin direction of device as output.
- ✓ Enable SPI in slave mode by setting SPE bit and clearing MSTR bit.

❖ SPI_Init function

```
void SPI_Init()                          /* SPI Initialize function */
{
    DDRB &= ~((1<<MOSI)|(1<<SCK)|(1<<SS));  /* Make MOSI, SCK, SS as
                                        input pins */
    DDRB |= (1<<MISO);                   /* Make MISO pin as
                                        output pin */
    SPCR = (1<<SPE);                     /* Enable SPI in slave mode */
}
```

Programming

- SPI Slave transmit steps
  - ✓ It has same function and steps as we do SPI Write in Master mode.
- SPI Slave Receive steps
  - ✓ Wait until SPIF becomes High.
  - ✓ Read received data from SPDR register.

❖ SPI_Receive function
Return value: it returns received char data type.

```
char SPI_Receive()            /* SPI Receive data function */
{
    while(!(SPSR & (1<<SPIF))); /* Wait till reception complete */
    return(SPDR);             /* Return received data */
}
```

Programming

- 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);
    }
}
```