8
SPI Port Based Serial Data Communication

8.1 Architecture of SPI Port

(1) Inside ATmega328P MCU of the UNO Board, there is a "SPI Port" (Fig-8.1) to exchange data in serial mode with another MCU (the NANO) or a sensor. SPI stands for "Synchronous Serial High Speed Full Duplex Peripheral Interface". There are four signal wires in the SPI Port.

SCK (Serial Clock),	MISO (Master-in Slave-out),
MOSI (Master-out Slave-in),	SS/ (Slave Select).

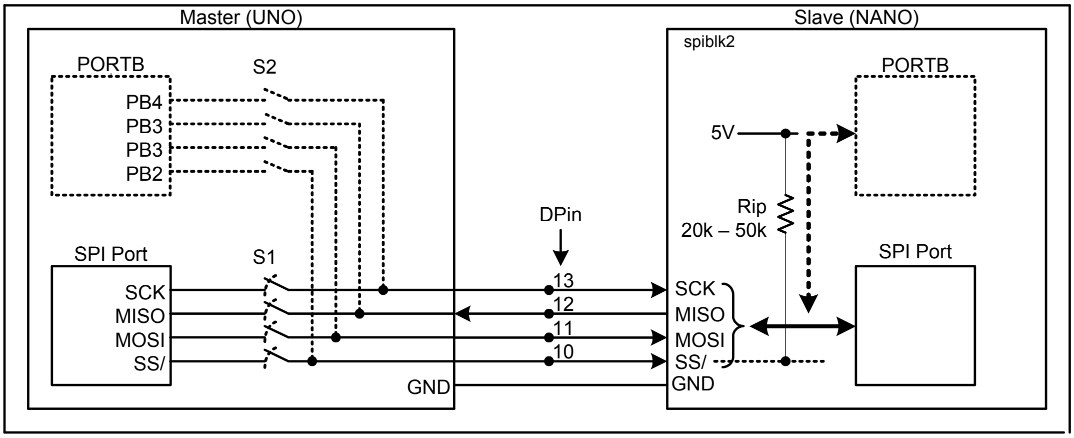

Figure-8.1: *Block diagram of the SPI Port of ATmega328P MCU*

(2) In SPI network, there is only one Master and one or more Slaves. The Slaves are selected one-after-another using SS/-signal. There is no address for a Slave like the I2C Bus.

(3) SPI stands for "Synchronous Serial High Speed Full Duplex Peripheral Interface".

i. It is a "synchronous" network; because, clock pulses are required to shift-in and shift-out data bits. It is the Master which always generates the clock pulses.

ii. It is a "high speed" network; because, it can handle data transfer rate as high as 8 Mbits/s compared to 3.4 Mbit/s of I2C bus, and 2 Mbit/s of UART Port.

iii. It is a "full duplex" network; because, data exchange between Master and Slave happens at the same time (Fig-8.2) compare to I2C bus which is “half duplex".

(4) [**SPI Port at Master Side]** IO lines (SS/, MOSI, MISO, SCK of Fig-8.1) of SPI Port are connected with external DPins (10, 11, 12, 13) when the following codes are included in sketch.

```
#include<SPI.h>     //contains ready-made routines, place in Global Space
SPI.begin();        //SPI Port is enabled. Place in setup() function.
```

SS (SS/) Dpin-10 output and default state is HIGH (SS is software name; SS/ is hardware name)
MOSI DPin-11 output and the default state is LOW
MISO DPin-12 input
SCK DPin-13 output and the default state is LOW

SS/ (Slave Select): This is an output line and the default initial value is HIGH. It is connected with the corresponding "slave select/cs(chip select)" line of the target Slave/sensor. The Master asserts LOW signal on this line to select the target Slave.

MOSI (Master-out Slave-in): It is an output line over which data bits from Master are shifted to Slave in synchronization with SCK (serial clock) signal (Fig-8.2). The default initial value of this line is LOW.

MISO (Master-in Slave-out): This is an input line and it carries data bits from Slave to Master in synchronization with SCK signal (Fig-7.2).

SCK (Serial Clock): This is an output line with default initial value of LOW. This line is connected with both SPDR Registers (Fig-8.2) of Master and Slave. Eight clock pulses are automatically generated on this line when 8-bit (1-byte) data are written on the SPDR Register of the Master. A SCK pulse simultaneously pushes-out a data bit from Master into Slave and pushes-in a data bit from Slave to Master. Thus, the SPI Port is circulating buffer type network.

(5) **[SPI Port at Slave Side]** IO lines (SS/, MOSI, MISO, SCK of Fig-8.1) of SPI Port are connected with external DPins (10, 11, 12, 13) when the following codes are included in sketch.

```
#include<SPI.h>//to get meanings for the symbolic names of: SS (SS/), MOSI, MISO, SCK

void setup()
{
    bitClear(SPCR, MSTR);   //now NANO is Slave MSTR = Master Bit = 1 means Master
    bitSet(SPCR, SPE);     //SPI Port is enabled
    pinMode(MISO, OUTPUT);
    pinMode(SS, INPUT_PULLUP);  // SS is the software name; SS/ is the hardware name
    SPI.attachInterrupt();   //or bitSet(SPCR, SPIE) to enable interrupt logic
}
```

(6) SPI Port is a byte (8-bit) oriented system. It means that data transfer/reception takes place 1-byte at a time. If there is 16-bit data to transfer, it has to be broken into two bytes before transfer.

(7) To transfer 8-bit data 8 SCK pulses are needed, which are automatically generated when 8-bit data is stored in the SPDR Register by executing one of the following two codes:

```
SPDR = 0x23;
Byte y = SPI.transfer(0x23);
```

(8) The **default SPI speed** of the UNO Board is **4 Mbits/s**. For demonstrative experiments, it is recommended to set the speed at 1 Mbits/s. The speed of the SPI Port could be set to a desired value by the following code:

```
SPI.setClockDivider(SPI_CLOCK_DIVX);//X=division factor of 16 MHz MCU clock. x=2,4,8,16,32,64,128
```

8.2 Programming of SPI Port

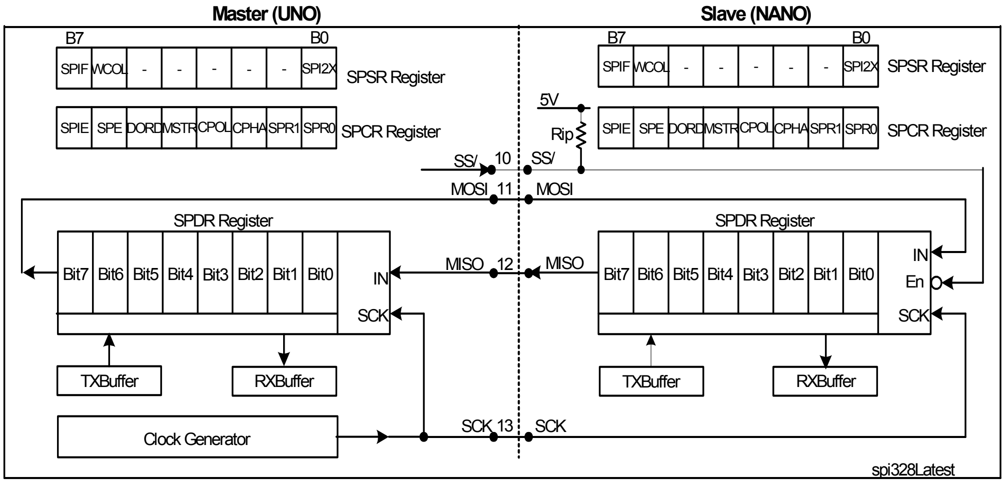

Figure-8.2: *Internal details f SPI Port*

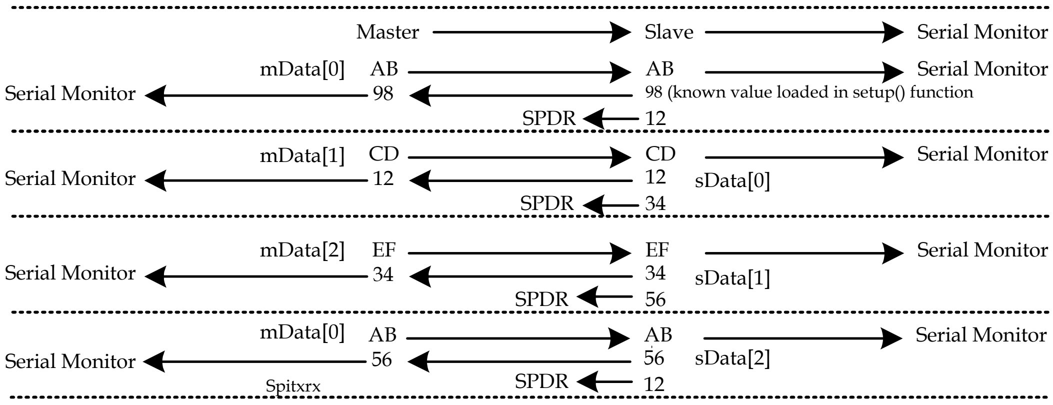

Figure-8.3: *Data exchange sequence between SPI-Master and SPI-Slave*

(1) Let us see referring to Fig-8.2 and 8.3 how data bytes 0xAB, 0xCD, and 0xEF enter into Slave from Master. We will also see what comes back to Master from Slave and in what sequence. Note that data exchange takes place via SPDR (SPI Data Register).

i. Assume that in `setup()` function of Slave, 0x98 has been initially stored in SPDR register.

ii. Master executes the following code:

```
byte y1 = SPI.transfer(0xAB);
Serial.println(y1, HEX);  //Master shows: 98
```

0xAB enters into Slave (Fig-8.2, 8.3) and then onto Serial Monitor. 0x98 enters into Master and then onto variable y1 and then onto Serial Monitor. Slave puts 0x12 into SPDR register which will go to Master in next bus cycle when Master will send 0xCD (Fig-8.3).

When a data byte enters into SPDR Register of Slave, the SPIF-bit of the SPSR Register of Slave becomes HIGH and interrupts the MCU (Fig-8.4). The MCU goes to ISR Routine (given below), collects data (0xAB) from SPDR and puts it into variable x1, sets flag to **true** to allow `loop()` function to show/print the value on Serial Monitor (after printing the value, the flag is reset). After that, the Slave puts data byte 0x12 into its SPDR Register.

```
ISR(SPI_STC_vect)  //STC = Serial Data Transfer Complete) vect for vector
{
     x1 = SPDR;     //read/brings data from SPDR register into variable x
     flag = true;   //indicates that MCU has come to this ISR due to interrupt
     SPDR = 0x12;  //data for Master
}

void loop()
{
     if(flag == true)
     {
          Serial.println(x1, HEX);   //slave shows: AB
          flag = false;
     }
}
```

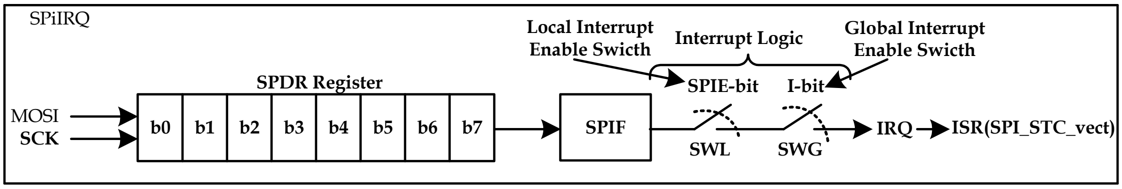

Figure-8.4: *Interrupt logic of SPI Port*

iii. Master executes the following code:

```
byte y2 = SPI.transfer(0xCD);
Serial.println(y2, HEX);  //Master shows: 12
```

0xCD enters into Slave (Fig-8.2, 8.3); 0x12 enters into Master and then onto Serial Monitor. Slave collect CD and shows Serial Monitor. After that, the Slave puts data byte 0x34 into its SPDR Register.

v. Master executes the following code:

```
byte y3 = SPI.transfer(0xEF);
Serial.println(y2, HEX);  //Master shows: 34
```

0xEF enters into Slave (Fig-8.2, 8.3); 0x34 enters into Master and then onto Serial Monitor. After that, the Slave puts data byte 0x56 into its SPDR Register.

iv. Master executes the following code:

```
byte y1 = SPI.transfer(0xAB);
Serial.println(y1, HEX);  //Master shows: 56
```

0xAB enters into Slave (Fig-8.2, 8.3); 0x56 enters into Master and then onto Serial Monitor. After that, the Slave puts data byte 0x12 into its SPDR Register.

v. The process keeps repeating,

(2) Sketch to exchange 3-byte data between Master-SPI and Slave-SPI

Master Sketch:

```
#include <SPI.h>
byte sendData[] = {0xAB, 0xCD, 0xEF};
byte recData[3];

void setup ()
{
  Serial.begin(9600);
  SPI.begin();  //default speed: 4 MHz
  digitalWrite(SS, LOW);   //Slave is selected
}

void loop()
{
  for (int i = 0; i < 3; i++)
  {
    recData[i] = SPI.transfer(sendData[i]);
    Serial.print("recData["); Serial.print(i); Serial.print("] = ");
    Serial.print(recData[i], HEX);
    Serial.print(" from Slave's sData[");
    Serial.print(i-1);
    Serial.println("] ");
  }

  long y = (long)recData[1] << 16 | (long)recData[2] << 8 | (long)recData[0];
  Serial.println(y, HEX);
  Serial.print("Data Received from Slave: ");
  Serial.println("========================");
  delay(3000);
}
```

Slave Sketch:

```
#include<SPI.h>  //include to get the meanings of SS, MISO, MOSI, SCK, etc.
int i = 0;
byte txData[] = {0x12, 0x34, 0x56};
byte rxData[3];
```

```
volatile bool flag = false;
bool flag1 = false;

void setup()
{
  Serial.begin(9600);
  pinMode(SS, INPUT_PULLUP);      // ensure SS stays high for now
  pinMode(MISO, OUTPUT);          //must do
  pinMode(MOSI, INPUT);     //always do
  pinMode(SCK, INPUT);     //always do
  bitSet(SPCR, SPE);      //SPI Port is enabled
  bitClear(SPCR,MSTR); //Arduino is Slave
  SPDR = 0x98;
  SPI.attachInterrupt();   //interrupt logic is enabled
}

void loop()
{
  if (flag == true)
  {
    rxData[i] = SPDR;
    Serial.print(rxData[i], HEX);
    SPDR = txData[i]; //places 0x12, then 0x34, then 0x56, then 0xAB
    i++;
    if (i == 3)          //3-byte data are sent
    {
      i = 0;             //array pointer is reset
      Serial.println();
    }
    flag = false;
  }
}

ISR(SPI_STC_vect)
{
  flag = true;
}
```

Output:

```
recData[0] = 56 from Slave's sData[-1 or 2] // -1 indicates data loaded in SPDR in the setup()
recData[1] = 12 from Slave's sData[0]
recData[2] = 34 from Slave's sData[1]
Data Received from Slave: 123456
=====================================
```

(3) Arduino Functions for SPI Port

	Function	
1	SPI.beginTransaction(SPISettings(speedMaximum, dataOrder, dataMode);	**Default settings:** **speedMaximu**: 4 Mbits/sec **dataOrder:** MSBFIRT

	a. speedMaximu: data transfer rate in bits/sec **b.** dataOrder: MSBFIRT or LSBFIRT **c.** dataMode: SPI_MODE0, SPI_MODE1, SPI_MODE2 or SPI_MODE3	**dataMode:** SPI_MODE0 SCK (clock) is at LOW state (CPOL = 0) Data is sampled at SCK's L-to-H (CPHA = 0)

2	SPI.begin();	Initializes the SPI bus;
		Makes SCK, MOSI, and SS outputs;
		Assets LOW on SCK and MOSI;
		Asserts HIGH on SS.

3	SPI.end();	Disables the SPI bus (leaving pin modes unchanged).

4	byte recByte = SPI.transfer(0xyy);
	byte recWord = SPI.transfer(0xyyzz);
		SPI transfer is based on a simultaneous send and receive: the received data is returned in recByte (or recWord).

5	Byte myData[= {0xAB, 0xCD, 0xEF};
	SPI.transfer(buffer, sizeof buffer);
		In case of transferring content of a buffer (array), the received data is stored in the buffer in-place (the old data is replaced with the data received).

6	SPI.setBitOrder();
	SPI.setClockDivider();
	SPI.setDataMode();
		These are not recommended to use in new projects.

8.3 Building Thermometer using LM35 sensor and exchanging data over SPI Port

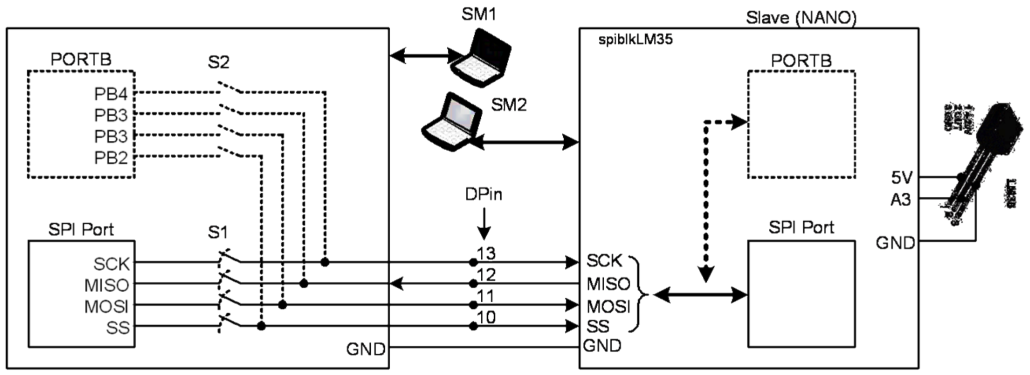

Figure-8.5: *Connection among Master-SPI, Slave-SPI, and LM35 sensor*

(1) Connect LM35 with UNO as per Fig-8.5. Create sketch to display temperature signal sensed by LM35 sensor on SM2 at 1-sec interval. Save program as P83LM35.

(2) Connect UNO and NANO using SPI Port as per Fig-8.5.

(3) Create sketch for UNO to receive temperature signal from NANO at 3-sec interval and show on SM1. Save program as P83M.

```
#include<SPI.h>
byte myData[] = {0x00, 0x00, 0x00, 0x00};
float myTemp;
```

```
unsigned long tempData;

void setup()
{
  Serial.begin(9600);
  SPI.begin();
  SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
  delay(100);
  digitalWrite(SS, LOW);  //Slave is selected
}

void loop()
{
  for (int i = 0; i < 4; i++)
  {
    myData[i] = SPI.transfer(myData[i]);
    delayMicroseconds(200);  //allow Slave to process the received data byte
    Serial.println(myData[i], HEX); //1M/0S; 2M/1S; 3M/2S; 0M/3S(MSB)
  }
  tempData = (unsigned long)myData[0] << 24 | (unsigned long)myData[3] << 16 |
             (unsigned long)myData[2] << 8 | (unsigned long)myData[1];
  toFloat();  //converting received binary32 bit t0 float
  Serial.print("Temperature received from Slave = ");
  Serial.print(myTemp, 1); Serial.println(" degC");//shows: Room Temp
  Serial.println("====================================");
  delay(3000);  //test interval
}

void toFloat()
{
  long *ptr;
  ptr = (long*)&myTemp;
  *ptr = tempData;
}
```

Or

```
void toFloat()
{
  static_assert(sizeof(tempData) == sizeof(myTemp), "");
  memcpy(&myTemp, &tempData, sizeof tempData);
}
```

(4) Create sketch for NANO to send temperature signal to UNO whenever such request comes from UNO. Save program as P83S.

```
#include<SPI.h>
byte myData[4];
int i = 0;
float myTemp;

void setup()
{
  Serial.begin(9600);
  analogReference(INTERNAL);
  pinMode(SS, INPUT_PULLUP);  // ensure SS stays high for now
  pinMode(MISO, OUTPUT);
  SPCR |= _BV(SPE);
  SPCR |= !(_BV(MSTR)); //Arduino is Slave
  SPI.attachInterrupt();   //interrupt logic is enabled
}

void loop()
{
  myTemp = (float)100 * (1.1 / 1023.0) * analogRead(A3);
  Serial.print("Room Temperature = ");
  Serial.print(myTemp, 1); Serial.println(" degC");
  toBytes();
  delay(2000);
}

ISR(SPI_STC_vect)
{
  SPDR = myData[i];
  i++;
  if (i == 4)     //4-byte data are sent
  {
    i = 0;          //array pointer is reset
  }
}

void toBytes()
{
  byte *ptr;
  ptr = (byte*)&myTemp;
  for (int i = 0; i < sizeof myTemp; i++)
  {
    myData[i] = *ptr;
    ptr++;
  }
}
```

or

```
void toBytes()
{
  static_assert(sizeof(myData) == sizeof(myTemp), "");
  memcpy(myData, &myTemp, sizeof myTemp);
}
```

8.4 Problems and Solutions