7.1 Introduction

(1) Inside the ATmega328P MCU of the UNO Board, there is a "SPI Port" (Fig-7.1) by which data exchange can take place between Master (UNO) and Slave (NANO) in serial fashion (bit-by-bit). In SPI network, there is only one Master and one or more Slaves; where, the Slaves are selected one-after-another as the need arises. There is no address for a Slave like the I2C Bus.

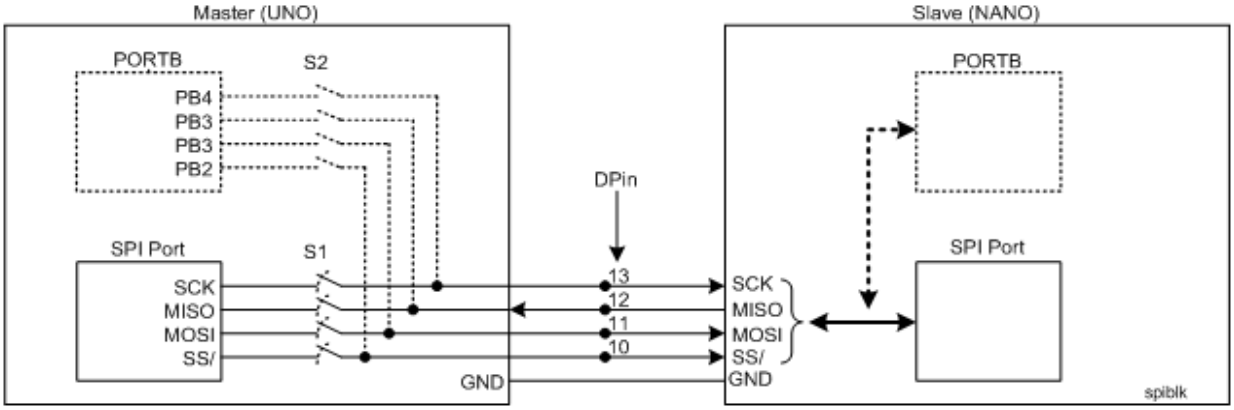

Figure-7.1: *Block diagram of the SPI Port structure of ATmega328P MCU of the Arduino UNO Board*

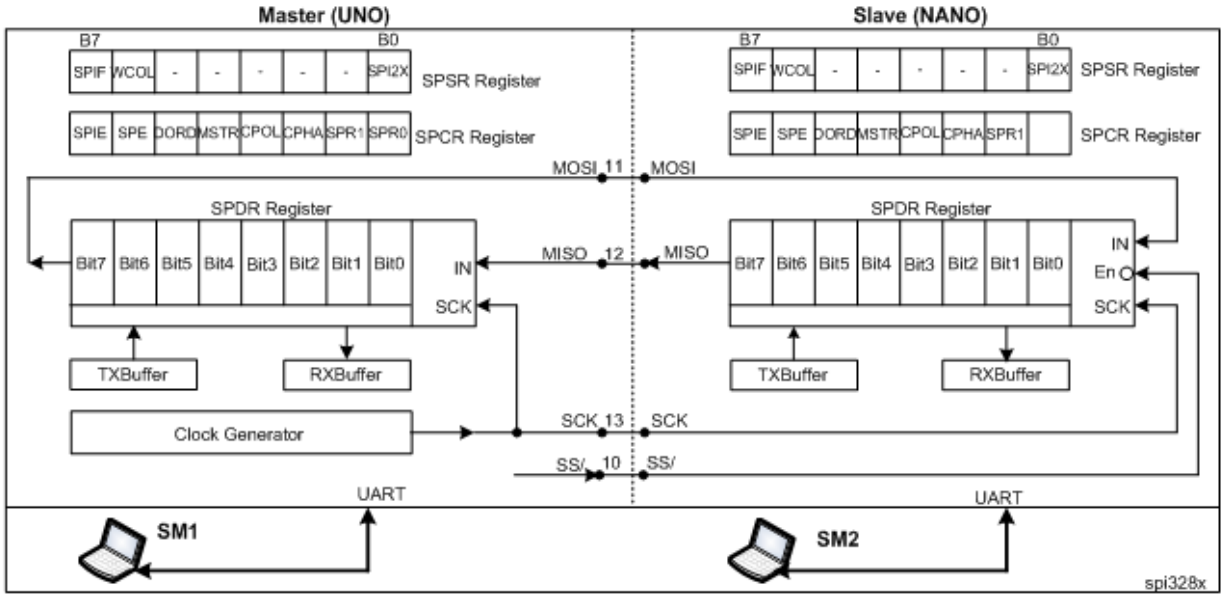

Figure-7.2: Expanded view of the SPI Port structure of ATmega328P MCU of the Arduino UNO Board

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

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

(a) 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.

(b) It is a "high speed" network; because, it can handle as high as 8 Mbits/s compared to I2C's 3.4 Mbit/s and UART's 2 Mbit/s.

(c) It is a "full duplex" network; because, data exchange between Master and Slave happens at the same time (Fig-7.2) compare to I2C's "half duplex".

(4) The default 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 = divide by 2, 4, 8, 16, 32, 64, 128 of 16 MHz
```

(5) The SPI Port is initialized/activated/created. The IO lines (SS/, MOSI, MISO, SCK of Fig-7.1) of the SPI Port are connected with external DPins (10, 11, 12, 13) when the following codes are included in the sketch.

```
#include<SPI.h>    //contains ready-made routines
SPI.begin();       //SPI Port is created and
```

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. 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-7.2). 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-7.2) 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.

(6) To deactivate the SPI Port and to disconnect its IO lines form the external DPins, the following code is executed.

```
SPI.end();
```

7.2 Master sends 1-byte data to Slave; Slave receives the data and shows it on its Serial Monitor

Assume that the Master wants to send 1-byte data (0x93) to the Slave. The following steps/tasks are to carried out:

(1) Keep the data byte (0x93 = 1001 0011) into SPDR Register of Master (Fig-7.2).

(2) Generate 8 SCK pulses to shift-out data bits (MSBit first) from Master and Shift-in into Slave. These 8 SCK pulses are automatically generated when the Master puts the said data byte (0x93) into its SPDR Register.

Notes: (a) Observe that when MSBit (1) of 0x93 shifts-out, the "MSBit of the SPDR Register of Slave" enters into Bit0 position of the "SPDR Register of Master". Here, we see that there is a simultaneous exchange of data between Master and Slave -- a process known as **Full Duplex.**

(b) Assume that the SPDR Register of Slave initially contains 0x87. Now at the end of 8-bit data exchange, SPDR Register of Master will hold .? (0x87).. and the SPDR Register of Slave will hold 0x93.

(c) Assume SPI speed is set 1 Mbits/s. How much time would be required for the exchange of 1-byte data of Note-b?

Ans: To exchange 8-bit data, 8 SCK pulses would be required. The duration of one SCK pulse is: $1/(1x10^6) = 1$ us. Therefore, the required time period is: 8x1 = 8 us.

(d) How does Slave know that the data byte 0x93 has arrived into its SPDR Register?

Ans: After the arrival of the data byte into the SPDR Register of the Slave, the SPIF-flag of the SPSR Register (Fig-7.2) assumes HIGH. Now the Slave can detect the arrival of the data in the following ways:

(i) Keep checking that the SPIF flag is HIGH and then bring the data byte from the SPDR Register into a variable **x** by executing the following codes:

```
while(bitRead(SPSR, SPIF) != HIGH)
{
    ;    //check and wait
}
bitSet(SPSR, SPIF);    //reset the SPIF flag
byte x = SPDR;  //bring the arrived data byte into variable x
```

(ii) If "SPI Interrupt Logic" of Slave is enabled by executing this code: **SPI.attachInterrupt();** in the setup() function, then the HIGH condition of SPIF flag will interrupt the MCU/Slave. As a result, the MCU/Slave will enter into the following ISR routine, read the data from its SPDR Register and save into variable **x**. (Because **Serial.print();** command is not allowed in an ISR routine, the Slave sets the **flag** variable in the ISR routine. This **flag** variable is later on used/tested in the loop() function for **true** and then the value of **x** is shown on the Serial Monitor of the Slave.)

```
ISR(SPI_STC_vect)  //SPI Serial Transfer Complete ; vect = vector
{
  x = SPDR;     //the 8-bit content of SPDR Register of Slave enters into variable x.
  flag = true;   //this flag will be used in loop() function to show x in Serial Monitor by Serial.print()
}
```

(e) How does Master know that the data byte 0x87 (Note-b) of the Slave has arrived into its SPDR Register?
Ans: The answer is similar to that of Note-d.

(3) Master Sketch:

```
#include<SPI.h> //Library file contains ready-made routines and the meanings of symbolic names

void setup()
{
    SPI.begin();      //SPI Port is initialized, created, activate, and connected with external DPins
    SPI.setClockDivider(SPI_CLOCK_DIV16); //SPI speed = 1 Mbit/s = 16 MHz /16
    //-------------------------------------------------------------------------------------------
    SPDR = 0x93;   //Master has put 0x93 into its SPDR Register, which automatically enters into Slave
}

void loop()
{

}
```

(4) Slave Sketch:

```
#include<SPI.h>  //SPI.h file is needed for the meanings of SS and MISO symbolic names
volatile byte x;
volatile bool flag = false;

void setup()
{
     Serial.begin(9600);   //UART Port is initialized, created, activated, and connected with DPins-0, 1
     //don't execute SPI.begin() in Slave sketch. Why? There is a problem. What is the problem? See Q21.
     //Initialize, create, and activate SPI Port for Slave; connect it with external Dpins- 10, 11, 12, 13
     pinMode(SS, INPUT_PULLUP);  //SS/-pin of Slave in made input with internal pull-up. This pi
     pinMode(MISO, OUTPUT);        //direction is: from Slave to Master
     SPCR |= _BV(SPE);                //SPI Port is enabled
     SPCR |= !(_BV(MSTR));          //NANO is made to work as Slave; MSTR-bit = HIGH means Master
     SPDR = 0x87;                       //initial value kept in the SPDR Register of Slave
     // SPCR |= _BV(SPIE);            //interrupt logic of Slave is enabled; _BV stands for BitValue
     SPI.attachInterrupt();           //alternate code at Arduino level to enable interrupt logic of Slave
}
```

```
void loop()
{
    if(flag == true)   //flag is true -- means the MCU/Slave visited ISR routine and has kept value in x
    {
        Serial.print(x, HEX);  //show the value of x onto Serial Monitor
        flag = false;              //reset the flag
    }
}

ISR(SPI_STC_vect)  //MCU/Slave comes into this ISR routine when a data byte arrives to Slave
{
   x = SPDR;     //bring data from SPDR Register into variable x
   flag = true;   //set flag to tell loop() function that variaable x contains data that come from Master
}
```

...see next post.

7.3 Master asks the Slave to send 2-byte data. Slave sends the data item to Master; Master receives the data item and then shows it on its Serial Monitor

Master Sketch:

```
#include<SPI.h>
byte myData[] = {0x00, 0x00};//,
void setup()
{
  Serial.begin(9600);
  SPI.begin();
  delay(100);
  SPI.setClockDivider(SPI_CLOCK_DIV16);//1 MBits/s
  //pinMode(SS, OUTPUT);
  digitalWrite(SS, LOW);  //Slave is selected
  //--------------------
}

void loop()
{
  for (int i = 0; i < 2; i++)
  {
    myData[i] = SPI.transfer(myData[i]);
    //delayMicroseconds(100); //allows Slave to process received byte
    Serial.println(myData[i], HEX);  //shows: 0xAB
  }
  int x = (int)myData[1] << 8 | (int)myData[0];
  Serial.println(x, HEX); //shows: 0x12345678
  Serial.println("======================");
  delay(1000);  //test interval
}
```

Slave Sketch:

```
#include<SPI.h>
int i = 0;
byte myData[] = {0x12, 0x34};//

void setup()
{
  Serial.begin(9600);
  SPI.setClockDivider(SPI_CLOCK_DIV16);//
  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()
{

}

ISR(SPI_STC_vect)
{
  SPDR = myData[i]; //places 0x12, then 0x34, then 0x56, then 0xAB
  i++;
  if (i == 2)     //2-byte data are sent
  {
    i = 0;          //array pointer is reset
  }
}
```

Master Serial Monitor:

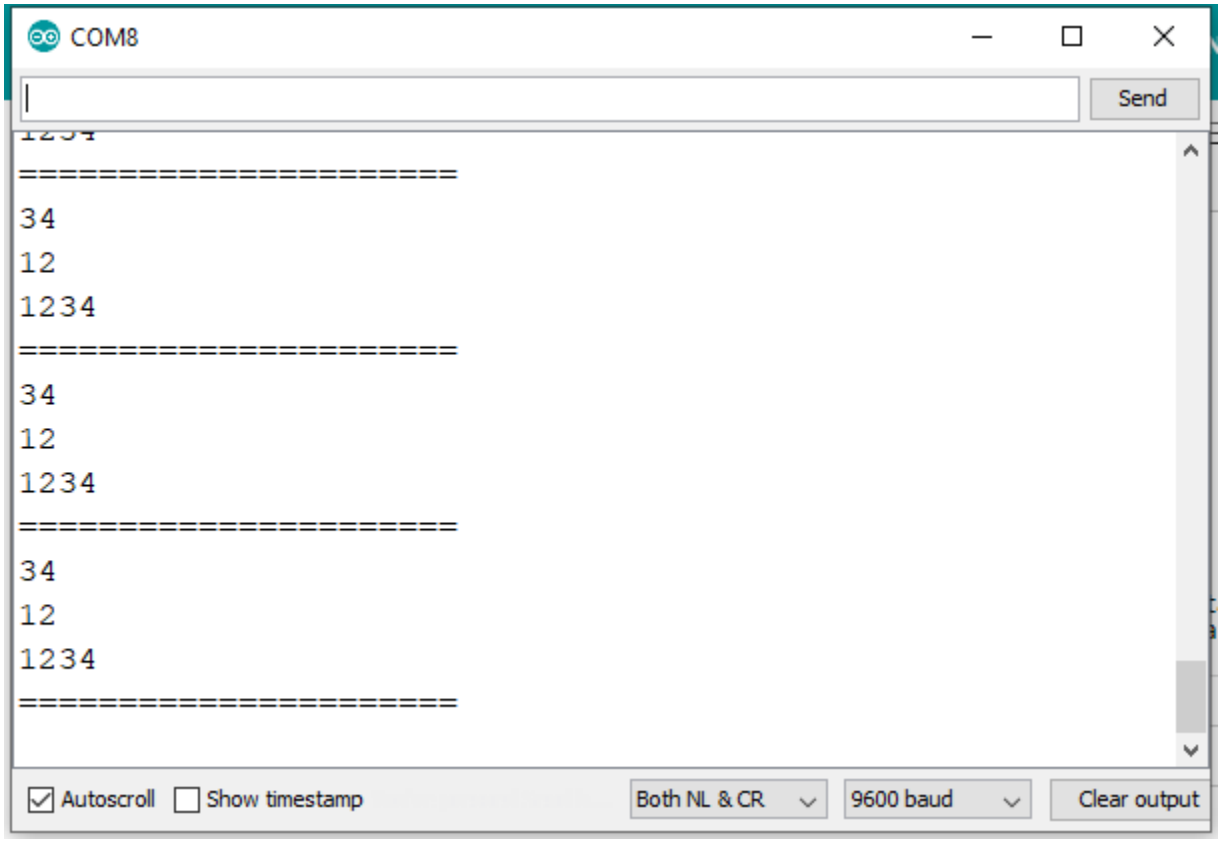

Fig-7.3:

7.4 Questions and Answers

Q1. Assume that the SPDR Registers of Master and Slave (Fig-7.2) contain 0x93 and 0x87 respectively. Write sketches for both Master and Slave for the exchange of data between them and show the received values in their respective Serial Monitors.

Ans: Master Sketch:

(1) Let us start with two blank functions -- setup() function and loop() function (Step-(9)). Leave some space on the top of setup() function to include global variables and Library file.

(2) Place the following code in the setup() function to connect the Serial Monitor.

```
Serial.begin(9600);
```

(3) Place the following codes in the setup() function to connect the SPI Port and to select the Slave.

```
SPI.begin();
digitalWrite(SS, LOW);   //
```

(4) Place the following code in the setup() function to set the SPI speed at 1 Mbits/sec.

```
SPI.setClockDivider(SPI_CLOCK_DIV16);  //SPI speed = 16 MHz/16 = 1Mbits/s
```

(5) Use the following code to place **SPI.h** Library at the global space in order to use ready-made routines/functions.

```
#include<SPI.h>
```

(6) (a) Place the following code (in the setup() function) to put data byte 0x93 into SPDR Register.

```
SPDR = 0x93;   //data byte will automatically enter into SPDR Register of Slave
```

(b) The above code could be replaced by the following Arduino Code:

```
SPI.transfer(0x93);    //valid only for Master
```

(7) (a) The data from Slave has probably arrived at Master. To ensure that the data has really arrived, we can check the value of SPIF flag of SPSR Register (Fig-7.2). So, the Master has to wait until SPIF flag goes to HIGH state. Place the following codes in the setup() function:

```
while(bitRead(SPSR, SPIF) != HIGH)
{
   ;    //null statement for waiting and checking that SPIF flag is HIGH
}
byte x = SPDR;    //data has arrived from Slave
bitSet(SPSR, SPIF);    //reset SPIF flag by putting HIGH at the SPIF position of the SPSR Register
```

The codes of Step-6(a) and 7(a) could be replaced by the following Arduino Code:

```
byte x = SPI.transfer(0x93);
```

(b) The Master could be known about the arrival of data byte from Slave by **interrupt process**. The interrupt process becomes enabled when the following code is placed in the setup() function.

```
SPI.attachInterrupt();
```

After the arrival of data from the Slave, the MCU/Master is automatically interrupted; the Master goes to the following ISR routine and saves the data in variable **x**. A flag is set to **true** in the ISR routine, which allows the loop() function to show the value of **x** in the Serial Monitor using Serial.print() command.

```
ISR(SPI_STC_vect)
{
   x = SPDR;    //read operation; data (that has come from Slave) is now in SPDR Register of Master
   flag = true
}
```

Include the above codes (the codes of ISR routine) at the end of loop() function of Master sketch.

(8) Place the following codes in the loop() function of the Blank Sketch of Step-(9) to show the value of variable **x** in Serial Monitor.

```
void loop()
{
   if(flag == true)
   {
      Serial.print(x, HEX);   // shows: 0x87 which has arrived from Slave in exchange of 0x93
      flag = false;              //flag is reset
   }
}
```

(9) (a) Blank Master Sketch//----------

```
void setup()
{

}

void loop()
{

}

ISR(SPI_STC_vect)
{
   x = SPDR;    //read operation; data (that has come from Slave) is now in SPDR Register of Master
   flag = true
}
//-----------------------------------------------------------------------------------------------------------
```

(b) Filled up Master Sketch

```
#include<SPI.h>

void setup()
{
  Serial.begin(9600);
  SPI.begin();
  delay(100);
  SPI.setClockDivider(SPI_CLOCK_DIV16);//
  //pinMode(SS, OUTPUT);
  digitalWrite(SS, LOW);
  //-------------------
}

void loop()
{
  SPDR = 0x93;
  while (bitRead(SPSR, SPIF) != HIGH)
  {
    ;
  }
  byte x = SPDR;
  bitSet(SPSR, SPIF);
  Serial.println(x, HEX); //shows: 0x87
  while (1);
}
```

Slave Sketch:

[b](1) Let us begin with two blank functions -- setup(0 function and loop() function (Step-(9)). Leave some space on the top of setup() function to include global variables and Library file.

(2) In the global space, place the following code to take advantage of the read-made functions of the SPI.h Library.

```
#include<SPI.h>
```

(3) In the setup() function, place the following code to connect the Serial Monitor.

```
Serial.begin(9600);
```

(4) To operate SPI Port, it is required to furnish the "SPI settings" (listed below) for both Master and Slave; where, the settings must be the same. Have we done these settings for Master? The answer is -- apparently no; but, these were (in fact) done under the umbrella of the execution of **SPI.begin();** code in the Master Sketch. For some reasons (to be explained later), the **SPI.begin();** code is not executed in the Slave Sketch; therefore, the "SPI Settings" for the Slave must be done manually. The settings are:

(a) Select option to enable "SPI Interrupt Logic",
(b) Connect/enable SPI Port with external DPins,
(c) Select which bit would be shifted out first,
(d) Select Master or Slave,
(e) Select Idle level of SCK line -- LOW or HIGH (this is CPOL = clock polarity),
(f) Select edge of SCK signal to sample data -- rising/falling edge (this is CPHA = clock phase),
(g) Select bit/s (speed),
//--
(h) Select "input mode with internal pull-up resistor' for SS/-pin,
(i) Select "output mode' for MISO signal, and
(j) Select "input mode' for SCK signal
//---

To accomplish the above "tasks/steps", we may include the following codes in the setup() function. The coding of "first 7 tasks/steps" involves the SPCR Register (Fig-7.2, 7.3) of the ATmega328P MCU of the UNO Board.

```
bitSet(SPCR, SPIE);       //SPIE = 1; SPI interrupt logic is enabled; Step-a
bitSet(SPCR, SPE);        //SPE =1;  SPI enable; Step-b
bitClear(SPCR, DORD);   //DORD = 0; Data bit shift order (MSBit first); Step-c
bitClear(SPCR, MSTR);   //MSTR = 0; NANO is a Slave; Step-d
bitClear(SPCR, CPOL);    //CPOL = 0; clock polarity (LOW); Step-e
bitClear(SPCR, CPHA);    //CPHA = 0; clock phase (leading edge sampling); Step-f
SPI.setClockDivider(SPI_CLOK_DIV16);   //1 Mbits/s ; Step-g
//-------------------------------------------------------------------------------------
pinMode(SS, INPUT_PULLUP);   //meaning of SS is known from SPI.h Library file; Step-h
pinMode(MISO, OUTPUT);         //meaning of MISO is known from SPI.h file; Step-i
pinMode(SCK, INPUT);             //meaning of SCK is known from SPI.h file; Step-j
//-------------------------------------------------------------------------------------
```

Note: The "first 6 tasks/steps" could be summarized by the following codes as some of the tasks are at the required states by default (Fig-7.4).

```
SPCR |= _BV(SPE);     //Step-b; _BV(SPE) stands for "Bit Value of SPE-bit is 1"
SPCR |= !(_BV(MSTR));      //Step-d
SPCR |= _BV(SPIE);         //Step-a
//  SPI.attachInterrupt();     //alternate Arduino code for Step-a
```

SPCR – SPI Control Register

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	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	

Figure-7.4: *Bit layout of SPCR Register*

... see next post/page.

(5) Place the following line in setup() function to store 0x87 (initial value) into SPDR Register.

```
SPDR = 0x87;
```

Question: Can the above code be replaced by the following code?

```
SPI.transfer(0x87);
```

(6) When data byte will arrive from Master, the Slave will be automatically interrupted; the Slave will enter into the following ISR routine; it will keep data in the variable **x** and then will set **flag** to true to allow the loop() function to show the value of **x** in the Serial Monitor.

```
ISR(SPI_STC_vect()
```

Place the above ISR routine along with necessary codes at the end of loop() function of Step-(9).

(7) Place the following codes in the loop() function to show the received value on Serial Monitor.

```
void loop()
{
   if(flag == true)
   {
      Serial.print(x, HEX);  //shows: 0x93
      flag = false;   //reset flag
   }
}
```

(8) Place the following declarations in the global area (at the top of setup() function).

```
volatile byte x;
volatile bool flag = false;
```

(9) (a) Blank Slave Sketch //----------

```
void setup()
{

}

void loop()
{

}

ISR(SPI_STC_vect)
{
   x = SPDR;    //read operation; data (that has come from Slave) is now in SPDR Register of Master
   flag = true
}
```

(b) Filled up Slave Sketch:

```
#include<SPI.h>

void setup()
{
  Serial.begin(9600);
  SPI.setClockDivider(SPI_CLOCK_DIV16);//
  pinMode(SS, INPUT_PULLUP);  // ensure SS stays high for now
  pinMode(MISO, OUTPUT);
  SPCR |= _BV(SPE);
  SPCR |= !(_BV(MSTR)); //Arduino is Slave
  //SPCR |= _BV(SPIE);
  //SPI.attachInterrupt();
  SPDR = 0x87
}

void loop()
{
  while (bitRead(SPSR, SPIF) != HIGH)
  {
    ;
  }
  byte x = SPDR;
  bitSet(SPSR, SPIF);
  Serial.println(x, HEX); //shows: 0x93
}
```

Program Execution:

(1) Upload Master Sketch and Slve Sketch.

(2) Press down and hold RESET buttons of both Arduinos.

(3) Release RESET button of Slave.

(4) Release RESET button of Master.

(5) Check that 0x93 has appeared on the Serial Monitor of Slave and 0x87 has appeared on the Serial Monitor of Master.

Q2. Write sketches for both Master and Slave of Q1 where arrived data would be read on interrupt process.

Q3. Write sketch for Master of Fig-7.2 to send the data item 0x3456 to the Slave. Also, write sketch for the Slave to receive the data sent from Master and show in the Serial Monitor of Slave.

Ans: Master Sketch Snippet

(1) Initialize everything as needed with SPI speed 1 Mbits/s.

(2) Include the following codes to send 0x3456 into Slave over two bus transactions; where, each transaction will send 1-byte data to the Slave and then wait for sufficient time to allow the Slave to process the received byte and then get ready for the next byte.

```
int x = 0x3456;
byte y1 = SPI.transfer(highByte(x));
delayMicroseconds(100); //wait for 50 us; transfer time 8 us + extra overhead time; see Slave sketch
//--------------------------------------
byte y2 = SPI.transfer(lowByte(x));   //goes 0x56
delayMicroseconds(100);
```

Complete Master Sketch:

```
#include<SPI.h>
int x = 0x3456;
void setup()
{
  Serial.begin(9600);
  SPI.begin();
  delay(100);
  SPI.setClockDivider(SPI_CLOCK_DIV16);//1 MBits/s
  //pinMode(SS, OUTPUT);
  digitalWrite(SS, LOW);  //Slave is selected
  //--------------------
  SPI.transfer(highByte(x));
  delayMicroseconds(100);
  SPI.transfer(lowByte(x));
  delayMicroseconds(100);
}

void loop()
{

}
```

Slave Sketch Snippet:

(1) Initialize everything as needed with SPI speed 1 Mbits/s.

(2) Include the following codes in the sketch to store the received bytes into myData[] array.

```
ISR(SPI_STC_vect)
{
   myData[i] = SPDR;   //store upper byte (0x34) and then lower byte (0x56)
   i++;
   if(i == 2)     //2 bytes have been received
```

```
   {
        flag = true;    //to allow loop(0 function to show the received value on Serial Monitor
        i = 0;            //reset array index
   }
}
```

(3) Include the following codes in the loop() function.

```
void loop()
{
   if(flag == true)
   {
        int y = myData[0]<<8 | myData[1]; //forming original 0x3456 data item
        Serial.print(y, HEX);     //shows: 0x3456
        flag = false;   //flag reset
   }
}
```

Complete Slave Sketch:

```
#include<SPI.h>
int i = 0;
volatile bool flag = false;

volatile byte myData[2];  //array to hold received bytes from Master

void setup()
{
  Serial.begin(9600);
  SPI.setClockDivider(SPI_CLOCK_DIV16);//
  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()
{
  if(flag == true)
  {
    int x = myData[0]<<8|myData[1];
    Serial.println(x, HEX);  //shows: 0x3456
    flag = false;
  }
}

ISR(SPI_STC_vect)
{
  myData[i] = SPDR; //received bytes are kept in array
  i++;
  if(i == 2)
  {
    flag = true;     //data received complete
    i = 0;          //array pointer is reset
  }
}
```

Program Execution:

(1) Upload Master sketch and Slave sketch.
(2) Press and hold RESET buttons on both Arduinos.
(3) Release the RESET button of Slave.
(4) Release the RESET button of Master.
(5) Check that Slave's Serial Monitor shows: 0x3456.
(6) Repeat the process going to Step-2.

Q4. Write expanded codes for the following single line code.

```
byte x = SPI.transfer(0x93);
```

Q5. Describe briefly -- what does happen when the Master of Fig-7.2 executes the following code?

```
byte x = SPI.transfer(0x93);
```

... to be continued.

Q6. Write codes to acquire 4-byte data (say: 0x123456AB) from a SPI Slave using SPI.transfer() instruction.
Ans:
Master Sketch:

```
#include<SPI.h>

void setup()
{
  Serial.begin(9600);
  SPI.begin();
  delay(100);
  SPI.setClockDivider(SPI_CLOCK_DIV16);//1 MBits/s
  //pinMode(SS, OUTPUT);
  digitalWrite(SS, LOW);  //Slave is selected
  //--------------------
}

void loop()
{
  byte x0 = SPI.transfer(0x00);//first brings garbage and then LSByte
  delayMicroseconds(100); //allows Slave to process received byte
  Serial.println(x0, HEX);  //shows: 0xAB
  //--------------------------
  byte x1 = SPI.transfer(0x00); //x1 = 0x12
  delayMicroseconds(100);
  Serial.println(x1, HEX);  //shows: 0x12
  //--------------------------
  byte x2 = SPI.transfer(0x00);  //x2 = 0x34
  delayMicroseconds(100);
  Serial.println(x2, HEX);    //shows: 0x34
  //--------------------------
  byte x3 = SPI.transfer(0x00);
  delayMicroseconds(100);
  Serial.println(x3, HEX); //x3 = 0x56
  //--------------------------
  long x = (long)x1<<24|(long)x2<<16|(long)x3<<8|(long)x0;
  Serial.println(x, HEX); //shows: 0x12345678
  Serial.println("======================");
  delay(1000);   //test interval
}
```

Slave Sketch:

```
#include<SPI.h>
int i = 0;
byte myData[] = {0x12, 0x34, 0x56, 0xAB};

void setup()
{
  Serial.begin(9600);
  SPI.setClockDivider(SPI_CLOCK_DIV16);//
  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()
{

}

ISR(SPI_STC_vect)
{
  SPDR = myData[i]; //places 0x12, then 0x34, then 0x56, then 0xAB
  i++;
  if (i == 4)     //4-byte data are sent
  {
    i = 0;          //array pointer is reset
  }
}
```

Master Serial Monitor

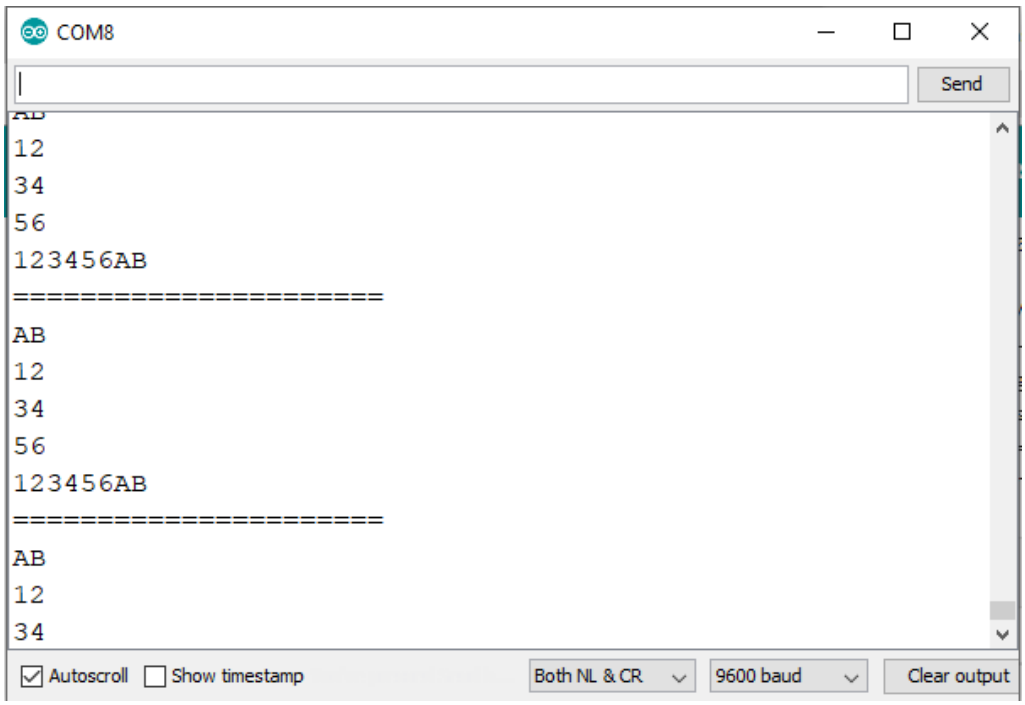

Fig-7.5:

Q7: Repeat Master Sketch of Q6 using for() loop. The Slave Sketch will remain the same.

Ans:

```
#include<SPI.h>
byte myData[] = {0x00, 0x00, 0x00, 0x00};
void setup()
{
  Serial.begin(9600);
  SPI.begin();
  delay(100);
  SPI.setClockDivider(SPI_CLOCK_DIV16);//1 MBits/s
  //pinMode(SS, OUTPUT);
  digitalWrite(SS, LOW);  //Slave is selected
  //--------------------
}

void loop()
{
  for (int i = 0; i < 4; i++)
  {
    myData[i] = SPI.transfer(myData[i]);
    delayMicroseconds(100); //allows Slave to process received byte
    Serial.println(myData[i], HEX);  //shows: 0xAB
  }
  long x = (long)myData[1] << 24 | (long)myData[2] << 16 | (long)myData[3] << 8 | (long)myData[0];
  Serial.println(x, HEX); //shows: 0x12345678
  Serial.println("======================");
  delay(1000);  //test interval
}
```

Q8. Write sketches for both Master and Slave of Fig-7.2 to acquire this float number: 32.67 from the SPI Slave.

Ans:

Master Sketch:

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

union myUData
{
  float x;
  byte myBytes[4];
};

myUData data;    //user defined data type myData is created using union keyword

void setup()
{
  Serial.begin(9600);
  SPI.begin();
  delay(100);
  SPI.setClockDivider(SPI_CLOCK_DIV16);//1 MBits/s
  //pinMode(SS, OUTPUT);
  digitalWrite(SS, LOW);  //Slave is selected
}

void loop()
{
  for (int i = 0; i < 4; i++)
  {
    myData[i] = SPI.transfer(myData[i]);
    delayMicroseconds(100); //allows Slave to process received byte
```

```
    Serial.println(myData[i], HEX);  //shows: 0xAB, 0x12, 0x34, 0x56
  }
  data.myBytes[3] = myData[0];
  data.myBytes[0] = myData[1];
  data.myBytes[1] = myData[2];
  data.myBytes[2] = myData[3];
  Serial.println(data.x, 2);//shows: 32.67
  Serial.println("======================");
  delay(1000);  //test interval
}
```

Slave Sketch:

```
#include<SPI.h>
int i = 0;
byte myData[] = {0x12, 0x34, 0x56, 0xAB};

union myUData
{
 float x;
 byte myBytes[4];
};
myUData data;

void setup()
{
 Serial.begin(9600);
 data.x = 32.67;
 SPI.setClockDivider(SPI_CLOCK_DIV16);//
 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()
{

}

ISR(SPI_STC_vect)
{
 SPDR = data.myBytes[i]; //places 0x12, then 0x34, then 0x56, then 0xAB
 i++;
 if (i == 4)    //4-byte data are sent
 {
   i = 0;        //array pointer is reset
 }
}
```

Master Serial Monitor:

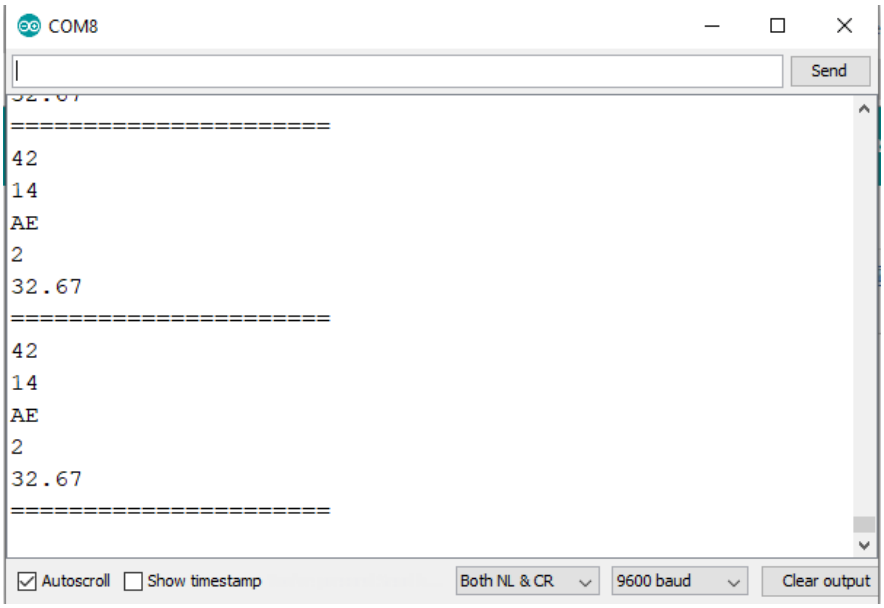

Fig-7.6:

Q9. Based on the following SPI network (Fig-7.7), collect temperature signal from the Slave and show it on SM1 at 2-sec interval.

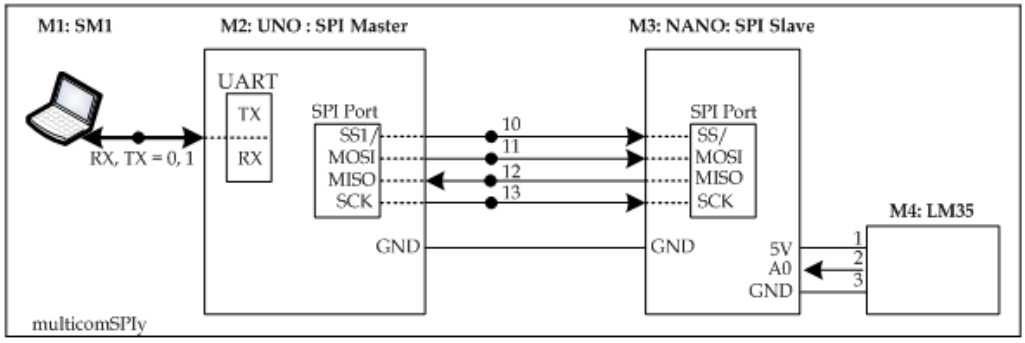

Figure-7.7: *SPI Port connection between UNO and NANO yo operate LM35 temperature sensor*

Master Sketch:

```
void setup()
{

}

void loop()
{

}
```

Slave Sketch:

```
#include<SPI.h>
int i = 0;
```

```
union myUData
{
  float x;
  byte myBytes[4];
};

myUData data;

void setup()
{
  Serial.begin(9600);
  analogReference(INTERNAL);
  SPI.setClockDivider(SPI_CLOCK_DIV16);//
  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()
{
  float myTemp = (float)100 * (1.1 / 1023.0) * analogRead(A0);
  data.x = myTemp;
  Serial.println(myTemp, 2);
  delay(2000);
}
ISR(SPI_STC_vect)
{
  SPDR = data.myBytes[i]; //places 0x12, then 0x34, then 0x56, then 0xAB
  i++;
  if (i == 4)     //4-byte data are sent
  {
    i = 0;          //array pointer is reset
  }
}
```

Master Serial Monitor:

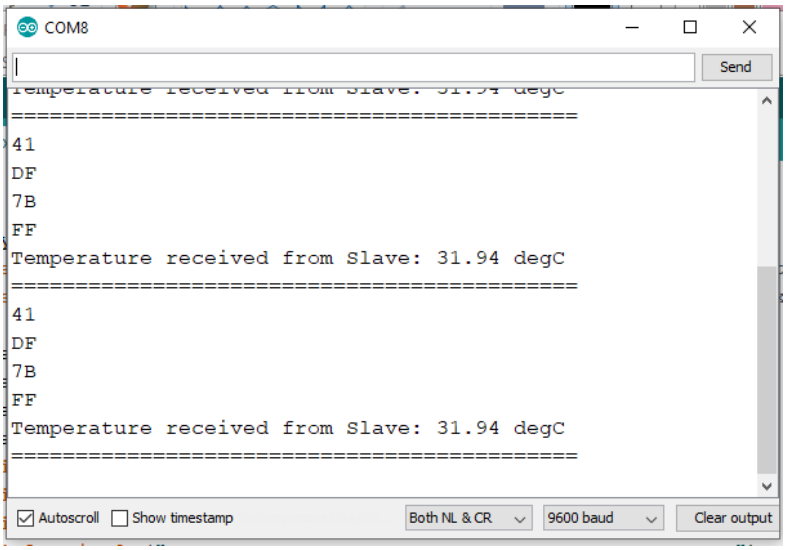

Fig-7.8: