Section-6 I2C Bus drivenSerial Communication

6.1 Introduction

(1) I2C (Inter-integrated Circuit, Fig-6.1) Bus is a 2-wire serial communication network among a Master and many Slaves. It is also known as TWI Bus (Two Wire Interface). The Master and the Slaves are usually placed on the same PCB within close proximity of about 10". I2C Bus is a byte oriented bus, and it always exchanges 1-byte (8-bit) data at a time.

(2) The Bus is composed of 2-wire (Fig-6.1)— SDA line which conveys data between Master and Slave; SCL line which conveys clock to shift-in data from Slave into Master and shift-out data from Master to Slave.

(3) I2C Bus can be operated as various speed grades such as: standard mode (100 kbit/s), full speed (400 kbit/s), fast mode (1 Mbit/s), and high speed (3.2 Mbit/s).

(4) It is called a Bus; because, a good number of MCUs/sensors/devices can be operated in parallel after assigning unique 7-bit address to each of the MCUs/sensors/devices. The I2C bus requires external pull-up resistors (2.2k/4.7k/10k).

(5) The I2C Bus is automatically created by borrowing PC4 and PC5 lines from Port-C when the user includes the following lines in the sketch. The default speed is 100 kbits/sec.

```
#include<Wire.h>
Wire.begin();
```

(6) In I2C Bus, aSlave is assigned a 7-bit address from 0 to 127 (0000000b – 1111110b) of which the first 8 addresses (0 to 7) are reserved. For a sensor, the address is usually permanently engraved within its I2C Logic; for MCU type slaves, the address is programmable and is assigned through program codes.

(7) When a data moves from Master to Slave, then this is a *write* operation and 0 (zero) is placed after the 7-bit address of the Slave in secret. When a data moves from Slave to Master, then this is a *read* operation and 1 (one) is placed after the 7-bit address of the Slave in secret.

6.2 Data Movement from UNO-Master to NANO-Slave

(1) Connection diagram between UNO and NANO using I2C Bus.

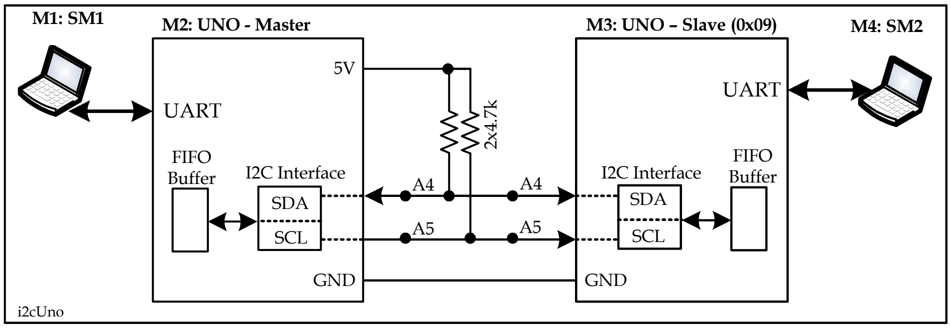

Figure-6.1: *I2C bus connection between UNO and NANO*

(2) The Master detects the presence of Slave (here: NANO with address 0001001) by making a roll call through the execution of the following codes:

```
Wire.beginTransmission(0b0001001);            //Slave Address + 0 (write mode) in silent
byte busStatus = Wire.endTransmission();      //
If(busStatus != 0x00)
{
     Serial.print("Slave is not found…!");
     while(1);      //wait for ever
}
```

When the address bits reach to the Slave, the Slave compares them with its own address. If a match occurs, the Slave creates an ACK pulse (by bringing down the SDA line for 1-bit period). The Master sensesthe ACK signal and generates a value of 0x00 for the busStatus signal.

(3) To send data byte 0x23 to the Slave, the Master executes the following codes:
```
Wire.beginTransmission(0b0001001);          //Slave Address + 0 (write mode) in silent
Wire.write(0x23);                           //data byte for Slave
Wire.endTransmission();     //
```

(4) When the data byte 0x23 reaches to the Slave, the Slave is interrupted; it goes to the TWISR (Interrupt Subroutine due to TWI Bus Interrupt) and saves the data into a FIFO Buffer. After that, the MCU moves to the following 'interrupt context' where the user brings out the data from FIFO Buffer, sets a flag, goes back to the calling function and then to the loop() function. In the loop() function, the Slave takes actions as needed depending on the value of flag. The codes are:
```
#include<Wire.h>
volatilebool flag1 = false;  //why is the keyword volatile?

void setup()
{
     Serial.begin(9600);
     Wire.begin(0b0001001);            //I2c Bus is created along with Slave address
     Wire.onReceive(receiveEvent);     //void receiveEvent(int howMany) is the interrupt context
}

void loop()
{
     if(flag1 == true);
     {
          Serial.print(x, HEX);    //shows: 23
          flag1 = false;           //ready for next cycle
     }
}

void onReceive(int howMany)  //howMany is always equal to number of bytes received from Master
{
     byte x  Wire.read();     //x contains 0x23
     flag1 = true;            //flag1 indicates that Slave has executed interrupt routine
}
```

(5) The Master/Slavecan execute the following code at the appropriate point of the sketch to know the number of data bytes that have been accumulated in the FIFO Buffer.
```
byte n = Wire.available();
```

6.3 Data Movement from NANO-Slave to UNO-Master

(1) When the Master wishes to read a data byte from a Slave, it executes the following code and then keeps waiting until requested number of data bytes arrive from the Slave:
```
Wire.requestFrom(0b0001001, 1);   //Slave address; 2ndarg is number of data bytes requested
```

In response, the following events occur:

(a) Slave is interrupted; it goes to TWISR and then goes to this interrupt context:
```
voidsendEvent()
{

}
```

(b) In the *sendEvent()* routine, the user puts data into Buffer for onward transmission to Master and then sets a flag.
```
voidsendEvent()
{
     Wire.write(0x45);   //data byte for Master
```

```
            flag1 = true;
        }
```

(c) The Slave goes to the calling function (the TWISR) and then goes to loop() function.

(2) Data has already arrived into the FIFO Buffer of Master from FIFO Buffer of Slave. Now, the Master executes the following codes to bring out the data byte from the Buffer.

```
byte x = Wire.read();
Serial.println(x, HEX);  //Serial Monitor shows: 45
```

6.4 Exercises

1 What does I2C stand for? Is there any alternate name for I2C? What is the distance covered by I2C Bus? Why do we call it I2C Bus instead of I2C Port?

(a) I2C stands for 'Inter Inter Circuit' or I^2C or Inter-integrated Circuit.

(b) The alternate name is **TWI (**Two Wire Interface) Bus.

(c) The I2C sensor/I2C device should be within 12" of the Controller.

(d) It is called Bus; because, we can make parallel operation of more than one I2C devices/sensors.

2 With a diagram, describe the mechanism of the formation and operation of I2C Bus.

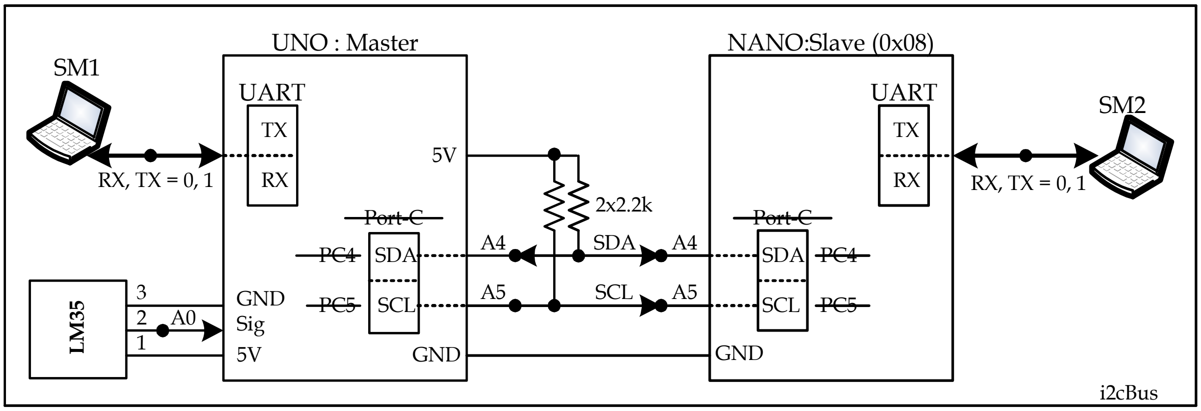

Figure-6.2: *I2C Bus connection between UNO-Master and NANO-Slave*

(1) There are two signal lines in the I2C Bus and these are: SDA Line and SCL Line. The SDA (Serial Data Line) carries serial data bit-by-bit between UNO and NANO. The SCL (Serial Clock Line) carries clock signal to shift-out data from UNO to NANO and shift-in data from NANO to UNO.

(2) The SDA Line is created by borrowing PC4-line of Port-C. The SCL is created by borrowing PC5-line of Port-C. Once the port lines are converted to I2C Bus, they are no longer good to work as IO lines of Port-C. This is the reason for deleting the words Port-C, PC4, and PC5 using horizontal lines across them. The SDA and SCL lines must be terminated to 5V by pull-up resistors of 2.2k or 4.7k or 10k.

(3) There are two lines in the I2C Bus for signaling purposes; but, we need these three lines to establish I2C Bus operation: SDA, SCL, and GND.

(4) In I2C Bus, one device (Microcontroller) works as a Master and all other devices work as Slaves. It is the Master which always generates the SCL pulses.

(5) In I2C Bus, every slave has a 7-bit address which is used to identify a particular slave in the bus. The range of the address is: 0x00 - 0x07, 0x08 – 0x7F; where, the 0x00 – 0x07 are the reserved addresses. In the above figure of Q2, we have assigned he address 0x08 (0001000) to the NANO Slave.

(6) How do we create an I2C Bus? The I2C Bus is created by including the following code lines in the sketch.

```
#include<Wire.h>
Wire.begin();
Wire.setClock(100000): //speed of data transfer : 100 000 bits/sec; this is also default speed
```

(7) In the I2C Bus, we can connect as many as 127 slaves at the same time. But, the Master can communicate with only one slave at a time. Therefore (before the operation of the I2C Bus begins) the Master must check that the target slave (with which it wants to communicate) is present. This is known as Roll Call.

(8) How does Master check the presence of a Slave?
The Master executes the following codes just for once to check the presence of the Slave in the I2C Bus.
```
Wire.beginTransmission(slaveAddres); //slaveAddres : 0001000 (7-bit)
bytebusStatus = Wire.endTransmission();
if(busStatus != 0x00)
{
     Serial.print("Slve is not present……….!");
     while(1);      //wait for evevr.
}
```

The Master sends the slaveAddres (0x08) to the slave NANO. When the slaveAddres sent by the Master matches with the slaveAddres (loaded into TWAR Register of Slave in the setup function), the Slave sends creates a low-going acknowledgment pulse (ACK) on SDA line. The Master senses this pulse and stores 0x00 in the variable named **busStatus**during the execution of **Wire.endTransmission ();** instruction.

(9) Now, the Master has found the Slave on the I2C Bus. The Master wants to send a data byte 0x35 to the Slave NANO. How does the Master do it? Let us note that the data exchange between Master and Slave is 1-byte at a time. This means that the I2C Bus is a byte oriented Bus.
Ans: The Master sends the data to the slave by executing the following codes:
```
Wire.beginTransmission(slaveAddres); //slaveAddres : 0001000 (7-bit)
Wire.write(0x35);  //this is the data byte that is going to Slave
Wire.endTransmission();
```
(10) The data byte of Step-9 has appeared to the NANO via its I2C Interface. Now, describing below of how the slave collects this data item and presents to the user.
Ans: When the data byte (or all the data bytes in the case of multiple data bytes) arrives at the Slave, the Slave is automatically interrupted. The slave goes to the following sub-program and reads the data byte from the FIFO into a user defined variable,
```
voidreceiveEvent(inthowMany)//variable howMany is always equal to the number of bytes received
{
     byte x = Wire.read();   //the data byte 0x35 has entered into variable x from the FIFO.
}
```

In order to make the above sub-program operational/accessible, it has to be declared under the setup() of the slave by executing the following single line code:
```
Wire.onReceive(receiveEvent);
```
3 To send a data byte (say, 0x35) to the slave, what codes does Master execute?
```
Wire.beginTransmission(slaveAddres);
Wire.write(0x35);     //data byte 35 is ready to be transmitted
bytebusStatus = Wire.endTransmission(); //slaveAddress and data byte are sent to Slave
```

Before sending the next data byte(say, 0x47), the Master much check that the slave has received the previous data byte (the 0x35). If the slave has really received the previous data byte, it would have sent the ACK signal to the Master. The Master would have received the ACK signal and stores a **status** value (0x00) into variable busStatus during the execution of **Wire.endTransmission();** instruction. Now, the Master can check the value of the variable busStatus by executing the following codes and then decide whether the next data byte (0x47) would be sent or not.
```
if(busStatus != 0x00)
{
     Serial.print("Slave has ………………...!");
     while(1);      //wait for evevr.
}
```
4 Fill up the gap of the argument field of the Serial.print() method of Q3.
5 Write codes to send data bytes 0x47 and 0x67 to NANO of Fig-2 using I2C Bus.
6 Write codes for Master to send the message Arduino to slave of Fig-2. Also, write codes for Slave to receive the message and show it on Serial Monitor.
Codes for Master:
```
Wire.beginTransmission(slaveAddres);
Wire.print("Arduino");
Wire.endTransmission();
```

Codes for Slave:
```
volatile bool flag1 = false;
Char myData[20] = "";
Wire.onReceive(receiveEvent);
void receiveEvent(int howMany)
{
     for(int i=0; i<howMany; i++)
     {
          myData[i] = Wire.read();
     }
     flag1 = true;
}
void loop()
{
     if(flag1 == true)
     {
          Serial.print(myData);
          char myData[20] = "";    //array reset
          flag1 = false;
     }
}
```
7 Organize the codes of Q6 into a sketch using setup() and loop() functions.

8 Given the integer variable ***int** x;*. Write codes to transmit the value of ***x*** to the Slave using I2C Bus. **Hints:** byte lowerByte = lowByte(x); and byte upperByte = highByte(x);.

9 You have two sensors connected in the I2C Bus; but, you don't know their addresses. Write program to find their addresses and print it on the Serial Monitor.
```
#include<Wire.h>
void setup()
{
     Serial.Begin(9600);
     Wire.begin();

     for(int i= 0; i<0x7F; i++)
     {
          Wire.beginTransmission(i);
          Byte BusStatus = Wire.endTransmission();
          If (busStatus == 0x00)
          {
               Serial.print("Slave found at address: 0x");
               Serial.println(i);
          }
     }
}

void loop()
{

}
```
10 Connect a 5k Pot with NANO as per following diagram. Write code for the Master to receive voltage sent by the Slave and display on the Serial Monitor of Master.

(1) At the NANO side, set the Pot to get 3.75V at the A1-pin. After digitization, it will appear as: (1023/5)*3.75 = 767 = 0x02FF. When we execute this instruction: unsigned int x = ananlogRead(A1);, the x will hold 0x02FF.

(2) At the UNO side: when the UNO wants to collect data from NANO (the Slave), the Master executes the following command:
```
Wire.requestFrom(slaveAddres, numberOfByteRequested);
⇒Wire.requestFrom(0x08, 2);   //2-byte data requested; because 0x02FF is 2-byte
```
(3) When the above command is executed, the Slave-NANO goes to the following sub-program and writes the data onto FIFO buffer for transmission to master.
```
voidsenEvent (inthowmany)
{
     Wire.write(lowByte(x));
     Wire.write(highByte(x));
}
```

(4) For the sendEvent() sub-program to become effective, it has to be declared in the setup() function of Slave by the following command:

```
Wire.onRequest(sendEvent);
```

(6) The Wire.requestFrom(0x08, 2); instruction waits until all data bytes have arrived to Master from the Slave. To bring data bytes from the Master FIFO to the user variables, the Master executes the following codes just after the requestFrom() command:

```
byte x0 = Wire.read(); //lower byte 0xFF is in x0
byte x1 = Wire.read(); //upper byte 0x02 is in x
//----------make them unsigned integer
Uint16_t  x = (uint16_t) x1<<8 |(uint16_t) x0
//--------- now convert 0x02FF back to 3.75 and show on Serial monitor-----
```

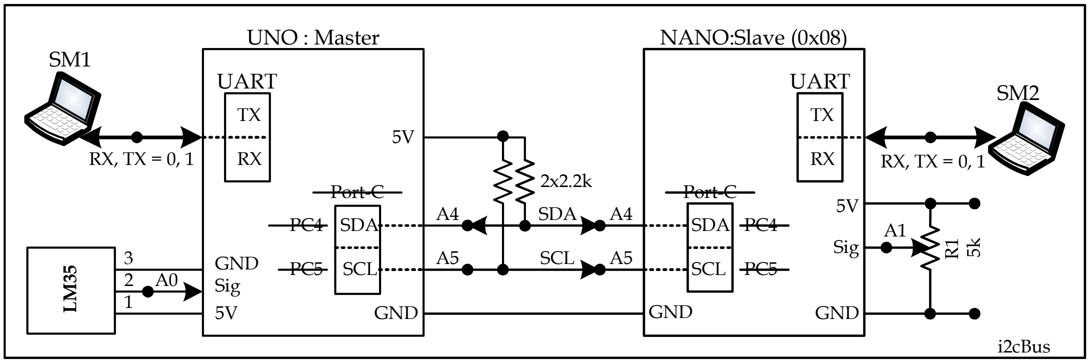

Figure-6.3: *I2C Bus connection between UNO-Master and NANO-Slave*

11 Create sketch to show the temperature signal of LM35 sensor of Fig-3 on SM2.

12 Write program to receive the Newline terminated string 8, 5000 from the InputBox of the Serial Monitor of UNO-Master and then send it to NANO-Slave using I2C Bus. After receiving the string, the Slave will present it in its Serial Monitor as two separate integer values with captions like:
Received integer-1 is: 8
Received integer-2 is: 5000

UNO-Master Codes:

```
#include<Wire.h>
voidsetup()
     {
  Wire.begin();
}

voidloop()
{
  Wire.beginTransmission(8);
  Wire.write('8');
  Wire.write(',');
  Wire.write('5');
  Wire.write('0');
  Wire.write('0');
  Wire.write('0');
  Wire.endTransmission();
  delay(1000);
}
```

NANO=Slave Codes:

```
char myData[20] = "";
volatile bool flag1 = false;
```

```
int x1, x2;

#include <Wire.h>

void setup()
{
  Wire.begin(8);
  Wire.onReceive(receiveEvent);
  Serial.begin(9600);
}

void loop()
{
  if (flag1 == true)
  {
    myData[1] = 0x00;  //null-byte in place of , (comma)
    x1 = atoi(myData);  //converting ASCII codes into integer using atoi() function
    Serial.println(x1, DEC);
    x2 = atoi(myData + 2);
    Serial.println(x2, DEC);
    flag1 = false;
  }
}
void receiveEvent(int howMany)
{
  for (int i = 0; i <howMany; i++)
  {
    myData[i] = Wire.read(); //6-byte ASCII data are in I2C Buffer; bring them out and save in
array
  }
  flag1 = true;
}
```

13 Write codes to convert the floating point number 32.65 into 32-bit (4-byte) number as per IEEE-754 (binary32) standard.

(1) When we declare this variable **float x = 24.93;**, then this (41C770A4) 32-bit value (4-byte) is automatically saved into the following four consecutive memory locations of the MCU.

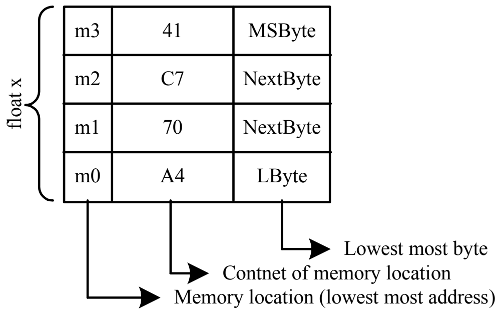

Figure-6.4: *Data structure showing the storage of 32-bit binarya32 formatted value of a floating point number*

(2) The value is computed based on a Rule known as IEEE-754 Standard or binary32 format.

(3) Now, our task is to write codes/know the ways of getting out the data bytes from these four unseen memory locations (Fig-4) and store them into four variables x0, x1, x2, and x3 where lower most data byte will be saved into variable x0, and so on.

(4) ***x*** is the name of the 'whole space' into which the 32-bit data is present. ***x*** is also the symbolic name of the 1st memory location (m0) of the 'array' composed of 'four (m0 – m3)' memory locations.

(5) Let us declare a pointer variable ***p***and store into it the address of the 1st memory location – m0.
byte *p; //p is a pointer variable; it points to a memory location which contains 1-byte data
p = (byte*) &x; //address of the 1st memory location (m0) is passed into pointer variable p. The casting (byte*) is made to force the system to deliver 1-byte data when p is used as a pointer variable.

(6) The complete codes:

```
byte *p
p = (byte*) &x;
byte x0 = *p;       // x0 = A4
p++;                // increment the pointer variable to point the next memory location (m1)
byte x1 = *p;       // x1 = 70
byte x2 = *(++p);   // x2 = C7
byte x3 = *(++p);   // x3 = 41
```

14 Write codes to convert (use union) the floating point number 24.93 into 32-bit (4-byte) number as per IEEE-754 (binary32) standard (rule).

```
union
{
     float x = 24.93;
     bytemyArray[4];    //myArray[0] will hold lower most byte of 32-bit vale = A4
} data;

byte x0 = data.myArray[0];    //A4
byte x1 = data.myArray[1];    //70
byte x2 = data.myArray[2];    //C7
byte x3 = data.myArray[3];    //41
```

15 Write codes to transmit the number 24.93 using I2C Bus to Slave NANO with slaveAddres 0x23.
Wire.beginTransmission(slaveAddres);

```
byte x0 = data.myArray[0];    //A4
byte x1 = data.myArray[1];    //70
byte x2 = data.myArray[2];    //C7
byte x3 = data.myArray[3];    //41
wire.endTransmission();
```

16 Repeat Q15 using *for()* loop to reduce the number of code lines.

17 When all the data bytes of Q15 have reached to the Slave, the Slave is automatically interrupted and saves the data bytes into FIFO buffer. The values can be brought out from the FIFO and save them into user-defined variables by entering into the following sub-program (it is an interrupt context; do not use print() command in this sub-program).

```
voidreceiveEvent(inthowMany) //the variable howMany is equal to the number of bytes so far received
{
     byte x0 = Wire.read();   //taken out lower most byte that has arrived first (A4)
     byte x1 = Wire.read();   //next data byte (70)
     byte x2 = Wire.read();   //next data byte (C7)
     byte x3 = Wire.read();   // next data byte (41)
}
```

18 Repeat Q17 using *for()* loop to reduce the number of code lines.

19 Write codes to retrieve the number 24.93 from the values of the variables x0 – x3 of Q17.

20 In I2C Bus, we have the address for I2CLCD as given 0x27. Write the address in bit form.

21 In I2C Bus, data exchange takes place one byte at a time. Write codes to transmit the number 0x1234 to Slave. Let us agree that we will always transmit lower byte if otherwise not stated.

22 Write codes for the Slave to read the received bytes of Q14 from the FIFO; reconstruct the original value 1234 and show it on the Serial Monitor.