# Question & Solve

## Chapter 5: UART

**1. Write codes to create the SUART (2, 3) Port.**

Code: [Select]

```
#include<SoftwareSerial.h>
SoftwareSerial SUART(2, 3);  //SRXD = DPin-2, STXD = DPin-3 ; S stands for Software
SUART.begin(9600);      //SUART Port works well at this bit speed = bit/sec
```

---

**2. Write codes to receive Newline terminated 12 (decimal) from InputBox of Serial Monitor of UNO, receive it by the UNO, send it to NANO using SUART Port and then NANO will receive the data, save at variable x.**

At UNO (Sender):

```
#include <SoftwareSerial.h>
SoftwareSerial SUART(2, 3);    //Rx(Dpin 2), Tx(Dpin 3)

void setup()
{
  Serial.begin(9600);    //Serial Monitor & MCU Communication starts
  SUART.begin(9600);     //Arduino Uno & Nano Communication starts
}

void loop()
{
  char x[20];

  while(Serial.available()>0)
  {
    byte m = Serial.readBytesUntil('\n', x,20);
    x[m] = '\0';
    Serial.println(x);
  }
  SUART.print(x);
  SUART.print("\n");
  delay(100);
}
```

AT Nano (Receiver):

```
#include <SoftwareSerial.h>
SoftwareSerial SUART(2, 3);  //Rx(Dpin 2), Tx(Dpin 3)

void setup()
{
Serial.begin(9600);     //Serial Monitor & MCU Communication starts
SUART.begin(9600);   //Arduino Uno & Nano Communication starts
delay(1000);
}

void loop()
{
  char x[20];
  while( SUART.available()>0 )  //Checking if serial communication is established
  {
    byte m = SUART.readBytesUntil('\n', x,20);
    x[m] = '\0';
    Serial.println(x); //here x is string, now wee need to convert x to integer

   byte value = (x[0]<<4) | (x[1]&0x0F); //string to binary==> 0001 0011
   value = ((value>>4)*0x0A) + (value & 0x0F);  //binary to Hex, now value = 0x0C
   Serial.println(value);

  }
}
```

**3. You are sending a string of 10 characters from the Input Box of the Serial Monitor. Write sketch to receive only 5 of them and show it on the Output Box of Serial Monitor.**

```
void setup()
{
  Serial.begin(9600);   //Serial Monitor & MCU Communication starts
}

void loop()
{
  char x[20];
  char send_data[6]="";


  while(Serial.available()>0 )
  {
    byte m = Serial.readBytesUntil('\n', x, 20);
    int i=0;
    while(i<5)
    {
      send_data[i]= x[i];
      i++;
    }
    send_data[i] = '\0';  //makes array to string
    Serial.println(send_data);
  }
  delay(100);
}
```

**4. Write meaning of the following code and then write expanded codes for it**

**byte m = Serial.readBytesUntil('\n', myData, 20);  //m = number of characters stored except '\n'**

Meaning:

Check if the arrived character is Newline character ('\n'); if not, save it into array myData[] and continue up to 20 characters; if it is a newline character, then stop receiving/storing characters. Do

not save the Newline character. Expanded code:

```
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  byte n = Serial.available();
  if(n != 0)
  {
    char x = Serial.read();
    if(x != '\n')
    {
      myData[i] = x;
      i++;
    }
    else
    {
      myData[i] = '\0'; //let there be a null byte as the last element in a char type array
      Serial.print(myData); //shows: 12
    }
  }
}
```

# Chapter 6: I2C Bus

**1.Write the codes that are to be executed at the both Master and Slave Sides to create the I2C Bus/Interface.**

(1) The I2C Bus is created at the Master side when the following codes are included in the Master sketch.
Code: [Select]

```
#include<Wire.h>            //contains ready-made functions/methods to do varieties of tasks
Wire.begin();                          //I2C Interface Module is connected with SDA and SCL lines
```

(2) The I2C Bus is created at the Slave side when the following codes are included in the Slave sketch.
Code: [Select]

```
#include<Wire.h>            //contains ready-made functions/methods to do varieties of tasks
Wire.begin(slaveAddress);          //I2C Interface Module is connected with SDA and SCL lines
```

**2. Assume that the Slave has initialized a variable with this string "Forum". Now, write necessary sketches so that the Master can issue command to the Slave to send the said string to Master and then Master will show the message in the OutputBox of its Serial Monitor.**

Code:

Master (Arduino Uno)

```
#include<Wire.h>

char x[10];   //so that 10 character can be saved
byte n;       //stores number of bytes coming

void setup()
{
 Wire.begin();
 Serial.begin(9600);

 n = Wire.requestFrom(0x52,5);  //requesting 0x52, to send 5 bytes,[Forum=5character]
}

void loop()
{
  if (Wire.available())
  {
    int i=0;
    while(i<n)  //n=5
    {
      x[i]= Wire.read();
      i++;
    }
    x[i] = '\0';  //to make "x" a string from array
    Serial.println(x);
  }
}
```

Slave (Arduino Nano)

```
#include<Wire.h>

void setup()
{
  Wire.begin(0x52);  //slave address to call by Master
  Serial.begin(9600);

  Wire.onRequest(sendEvent);
}

void loop()
{
  ;
}

void sendEvent()
{
  char m[]="Forum";  //sending messege
  int l = sizeof(m);
  Wire.write(m, l);
}
```

**3. Write sketch to check the presence of Slave of Fig-2 at address 0x52 and then show the following message on the Serial Monitor of Master. Your searching sketch should cover the address range: 0x08 to 0x7E. " Slave Found at address: 0x52 "**

Master (Arduino Uno)

```
#include<Wire.h>

byte start_address= 0x08;
byte end_address= 0x7E;
void setup()
{
 Wire.begin();
 Serial.begin(9600);

 while (start_address <= end_address)
  {
    byte n = Wire.requestFrom(start_address, 1);
    if ( n >0 )
    {
        Serial.print("Salve found at address : 0x");
        Serial.println(start_address, HEX);
        break;
    }
    start_address++;
  }
 if(start_address > end_address)
 {
  Serial.print("No slave found");
 }
}

void loop()
{
;
}
```

Slave (Arduino Nano)

```
#include<Wire.h>

void setup()
{
  Wire.begin(0x52);  //slave address to call by Master
  Serial.begin(9600);

  Wire.onRequest(sendEvent);
}

void loop()
{
  ;
}

void sendEvent()
{
 Wire.write(0x35);
}
```

# Chapter 7: SPI

**1.Write codes to create SPI Ports for both Master and Slave.**

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

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

**2.Write sketches for both Master and Slave to send 0x1234 from Master; the Slave will receive the data and will be displayed on the Serial Monitor of Slave.**

Master (Arduino Uno):

```
#include<SPI.h>

byte sendData[]={0x12, 0x34};

void setup()
{
 Serial.begin(9600);
 SPI.begin();
 delay(100);

 SPI.setClockDivider(SPI_CLOCK_DIV16);
 digitalWrite(SS, LOW);
}

void loop()
{
  SPI.transfer(sendData[0]);
  delay(100);
  SPI.transfer(sendData[1]);
  delay(1000);
}
```

Slave (Arduino Nano)

```
#include<SPI.h>

int i=0;
volatile bool flag=true;

byte receiveData[]={0x00, 0x00};

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

  pinMode(SS, INPUT_PULLUP);
  pinMode(MISO, OUTPUT);

  SPCR |= _BV(SPE);
  SPCR |= !(_BV(MSTR));
  SPI.attachInterrupt();

}

void loop()
{
  if(flag == true)
  {
    Serial.print("0x");
    Serial.print(receiveData[0], HEX);
    Serial.println(receiveData[1], HEX);  //total output= 0x1234
  }
  flag = false;
  delay(100);
}

ISR(SPI_STC_vect)
{
  receiveData[i]= SPDR;
  i++;
  if(i==2)
  {
    i=0;
    flag=true;
  }
}

```

**3.When does SPIF flag get set (assume HIGH state)?**

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:
Code: [Select]

```
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.)
Code: [Select]

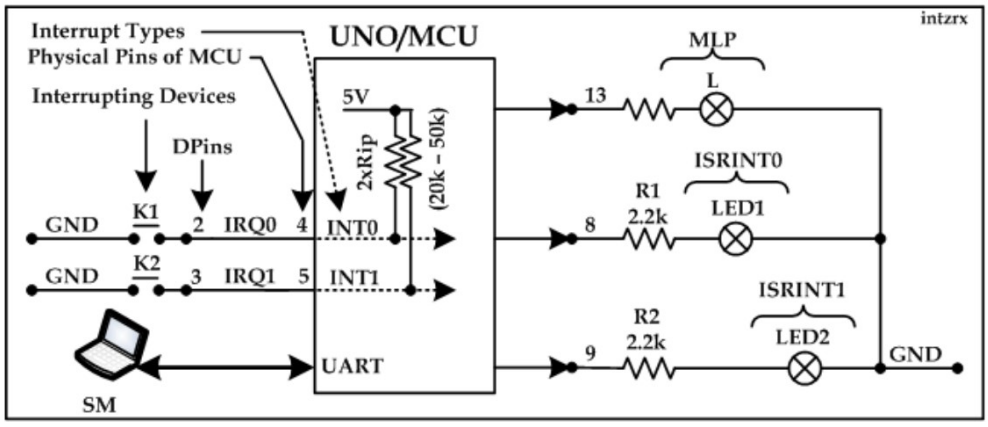
```
ISR(SPI_STC_vect)  //SPI Serial Transfer Complete ; vect = vector
{
  x = SPDR;     //the 8-bit content of SPDR Register of Slave enters into variable x.
  flag = true;   //this flag will be used in loop() function to show x in Serial Monitor by Serial.print()
}
```

# Chapter 8: Interrupting

**1.Interrupting the Micro Controller-**

Interrupting the MCU means telling it to suspend what it has been doing.

- The MCU is continuously blinking L at 1-sec interval. "Blinking L" is a job that is known as "Main Line Program (MLP).
- Now, ask the MCU by pushing the external switch K1 to suspend the MLP of Step-1 and then blink LED1 only for 3 times at 2-sec interval and then resume the MLP. "Blinking LED1" is a "side job" which is known as "Interrupt Service Routine (ISR)".
- The symbolic name of the ISR is: ISRINT0.



**2. Write Sketch, When MCU interrupt by K1, Lm35 will read data and show on Serial monitor-**

Let's assume K1 is installed on Dpin-2 to interrupt MCU. There will be a interrupt function "void ISRINTZ()" to let know MCU that an Interrupt occurred.

Code:

```
byte counter = 3;
volatile bool flag = false;

void setup()
{
  Serial.begin(9600);
  pinMode(A0, INPUT);
  pinMode(2, INPUT_PULLUP);    //K1 installed

  attachInterrupt(digitalPinToInterrupt(2), ISRINTZ, LOW);  // for interrupting
  analogReference(INTERNAL);  //for LM35
}
void loop()
{
  if( flag == true )
          {
  float data=analogRead(A0);    //read from sensor
  float t=100*(1.1/1024)*data;  //LM35 equation
  Serial.println(t, 2);  //shows only 2 digits after floating point
         }
flag = false;
}
void ISRINTZ()
{
   flag = true;  //whenever K1 is pressed this function will run first then void loop().
}
```

**3. Describe how does the IRQ-signal at DPin-2 cause the MCU to jump at the ISR routine named ISRINT0()-**

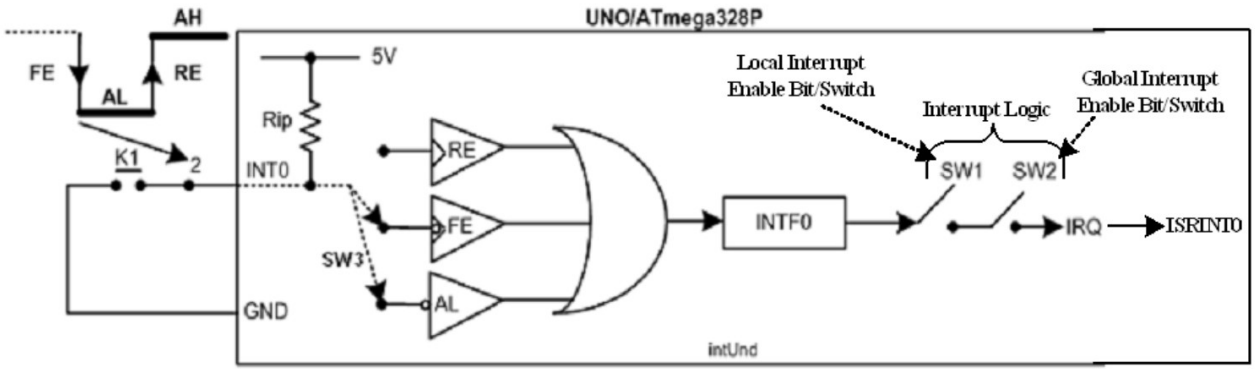



***Figure-8.2:*** *Interrupt trigger levels of ATmega328P MCU*

- At the presence of any event (LWO or FE or RE) at DPin-2, the INTF0 bit/flag of the MCU assumes HIGH state. This flag generates interrupt signal (IRQ) for the MCU provided and SW2 are put into closed conditions (enabled states) in the setup() function. SWI and SW2 together makes "something" what is called the "interrupt logic".

- SWI and SW2 becomes closed means "interrupt logic" becomes enabled when this code: " attachinterrupt (argl, arg2, arg3); " ,is executed in the setup() function.

- We observe that the ATmega328P of the UNO Board has two physical pins to receive interrupt signals from external sources. The symbolic names of these two pins ate: INTO and INTl which are connected with DPin-2 and 3 respectively.

- After interruption, the MCU will suspend the MLP and then will jump to ISR routine; therefore, a code should be executed in the setup() function to establish a relation/connection between DPin-2 (IRQO comes to INTO-pin of MCU via this DPin-2) and ISRINTO routine. The following is the code that makes the said relation/connection.

  attachInterrupt(digitalPinToInterrupt(DPin), ISRName, TriggerLevel);

  ==> attachInterrupt(digitalPinToInterrupt(2), ISRINT0, LOW); //for INT0 interrupt

# Chapter 9: Timer

**1.Compose 9 sentences to express the key information contained in Fig-6.**

- Within ATmega328P MCU of the Arduino UNO Board, there are three "Timer/Counter Modules/Registers" (Fig-9.1). These are:
  a) 8-bit TC0 (Timer/Counter 0) Module/Register. It is also Known as TCNT0 (Timer/Counter 0).
  b) 16-bit TC1 (Timer/Counter 1) Module/Register. It is also Known as TCNT1 (Timer/Counter 0).
  c) 8-bit TC2 (Timer/Counter 2) Module/Register. It is also Known as TCNT2 (Timer/Counter 2).

- TCNT1 is a 16-bit Register/Module. It is composed of two parts: TCNT1H (high byte of TCNT1) and TCNT1L (low byte of TCNT1). In "Normal Mode" operation, the TCNT1 always counts in the upward direction; this means that TCNT1 is an "up counter" register.

- TCNT1 can receive the counting/driving pulses (clkTC1) from the following two sources:
  a) From internal "16 MHz Oscillator"
  b) From external source via DPin-5

- When TCNT1 receives its driving clkTC1 from internal oscillator, it is said that TCNT1 is working as Timer-1.
- When TCNT1 receives its driving clkTC1 from external source, it is said that TCNT1 is working as Counter-1.
- TCNT1 begins the counting of clkTC1 pulses from 0x0000 (inital count or BOTTOM) and goes upto 0xFFFF (final or MAXimum count) and then again to 0x0000 after counting one.
- The occurrence of the over-flow event can be known in two ways:
  a. Polling Process
  b. Interrupt Process

- Storing 0x2356 into TCNTl Register-
  TCNT1H = 0x23;
  TCNT1L =0x56

- Data read sequence from TCNTl Register
  byte x1=TCNT1H
  byte x2=TCNT1L

**2.Write Pseudo Codes/Text Codes to initialize TC2 so that overflow event occurs at the elapse of 500 ms time period.**

**(Not Sure)**

We know,
Total clock pulse counts of clkTC2 = 2^8 + 1 //clkTC2 = 8 bit
= 257
frequency = 16 MHz/divission factor of TC2 Clock Prescaler
= 16 MHZ/ 1024
= 15625 HZ

| | | | |
|---|---|---|---|
| in | 1 s | the clock gives | 15625 pulses |
| in | (500/1000) s | the clock gives | 15625* (500/1000) pulses |
| | | | =7812.5 pulses |

So, At given 500ms, the new actual counts = 7812.5

We know,
preset count + new actual count = total counts
preset count = total counts - new actual counts
= 257 - 7812.5
= -7555.5 in decimal
= -1D83.8 in Hex (Ans)

# Chapter 10: PWM

**1. Generate 490 Hz PWM signal at DPin-3. the brightness of LEDI will be gradually changing as we gradually rotate/turn the pot (potentiometer) connected at Ch0 of the ADC via AO-pin.**

```
void setup()
{
 Serial.begin(9600);
 pinMode(A0, INPUT);
 pinMode(3, OUTPUT);
}

void loop()
{
  float pot = analogRead(A0);
  float pwm = map(pot, 0,1023, 0,255);
  analogWrite(3, pwm);
}
```

**2. Write a single line code for the 4-line codes of the loop() function of Question-1**

➔ analogWrite(3, map( analogRead(A0), 0,1023, 0,255) );

# Ch-11: Peripheral Controllers and Devices

**1. Write sketch to show the temperature signal of the LM35 sensor at 2-sec interval on the display unit of the Fig-11.1 using Library Functions/Methods. The prototypes and meanings of the methods could be found in the attached Library file (SevSeg-master.zip). Extract the zip file and then get the syntax rules and meaning of the methods. (You may take take help of the given sketch).**

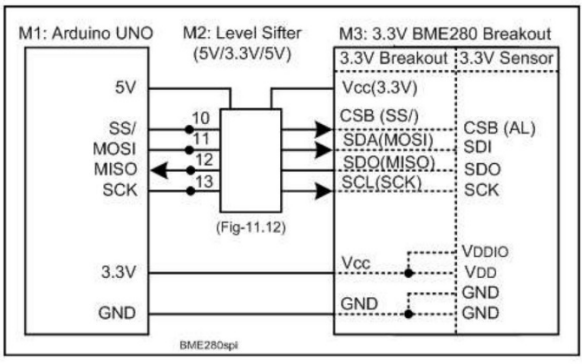
Code:

```
#include "SevSeg.h"
SevSeg sevseg;
byte numDigits = 4;
byte digitPins[] = {A0, A1, A2, A3};//DP0, DP1, DP2, DP3
byte segmentPins[] = {8, 9, 10, 11, 12, 13, 6, 7};//a, b, c, ..., g, p from left
bool resistorsOnSegments = false;
byte hardwareConfig = COMMON_CATHODE;

void setup()
{
  Serial.begin(9600);
  sevseg.begin(hardwareConfig, numDigits, digitPins, segmentPins, resistorsOnSegments);
  sevseg.setBrightness(30);
  analogReference(INTERNAL);
}

void loop()
{
  unsigned long prMillis = millis();
  while (millis() - prMillis < 2000)//temp refreshed at 2-sec interval
  {
    sevseg.refreshDisplay();
  }

  float temp = 100 * ( 1.1 / 1023) * analogRead(A4);//31.76
  sevseg.setNumber(temp, 2);//decimal before 2-digit from right
}
```

# Ch-12: Sensors

**1. Draw 4-wire SPI connection between 3.3V BME280 Sensor Breakout Board of Fig-12.13 and the UNO. The Breakout Board is without on-board Level Sifters and Voltage Regulator**

Solution:



# Ch-13: Taxi

**1. Write the functions of the buttons: RST, HIRED, TO PAY, and LED TEST.**

RESET: The driver pushes this button to bring 0.00 into DDM, 0.00 into DFM, and 0.00 into DWM.

HIRED: The driver pushes this button when the Taxicab is hired. As a result, o.00 comes into DDM, 40.00 (as Flat Fare) on DFM, and 0.00 into DWM. This button will work only for once during the whole journey and after the rest button that brings Tk 0.00 in DFM.

TOPAY: the driver pushes this button at the end of journey. As a result, all activities of the Taxi Meter gets frozen. The display reading do not change. The passenger records the distance traveled, fare charged, and the waiting time. This button will work only when the taxicab is in "waiting state".

LED TEST: (Unknown)

**2. Sorry vai, eita possible na. Onek long**