DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

Fr. Conceicao Rodrigues College of

Engineering Bandra (W) Mumbai – 400050

Name of the student: Arpita Apurba Kar

Roll No: 9073

Semester: VI

Subject: Embedded Systems and Real time Operating Systems.

Subject Code: ECC 601

Academic Year: January to April, 2022

Index

TE Electronics & Computer Science R2019 Semester VI

Subject: Embedded systems and real time operating systems Lab

Subject Code: ECL601

1. Display Interfacing

Course, Subject & Experiment Details

Academic year	**2021 – 2022**	**Estimated Time**	**02 Hours**
Course	**T.E. (ECS)**	**Subject Name**	**Embedded systems and RTOS**
Semester	**VI**	**Chapter Title**	**Display interfacing**
Experiment Type	**Coding**	**Subject Code**	**ECC 601**

Aim & Objective of Experiment

1. To write and execute an assembly language program to interface an alphanumeric LCD to the 8051 (Using KEIL Micro-Vision)

2. To design the system (using Proteus VSM) and show simulation results.

Theory:

Liquid Crystal Display (LCD) consists of rod-shaped tiny molecules sandwiched between a flat piece of glass and an opaque substrate. These rod-shaped molecules in between the plates align into two different physical positions based on the electric charge applied to them. When electric charge is applied they align to block the light entering through them, whereas when no-charge is applied they become transparent. Light passing through makes the desired images appear. This is the basic concept behind LCD displays. LCDs are most commonly used because of their advantages over other display technologies. They are thin and flat and consume very small amount of power compared to LED displays and cathode ray tubes (CRTs).

Advantages:

· Consumes less power and generates less heat.
· Saves lot of space compared picture tubes due to LCD's flatness.
· Due to less weight and flatness LCDs are highly portable.
· No flicker and less screen glare in LCDs to reduce eyestrain.

Drawbacks:

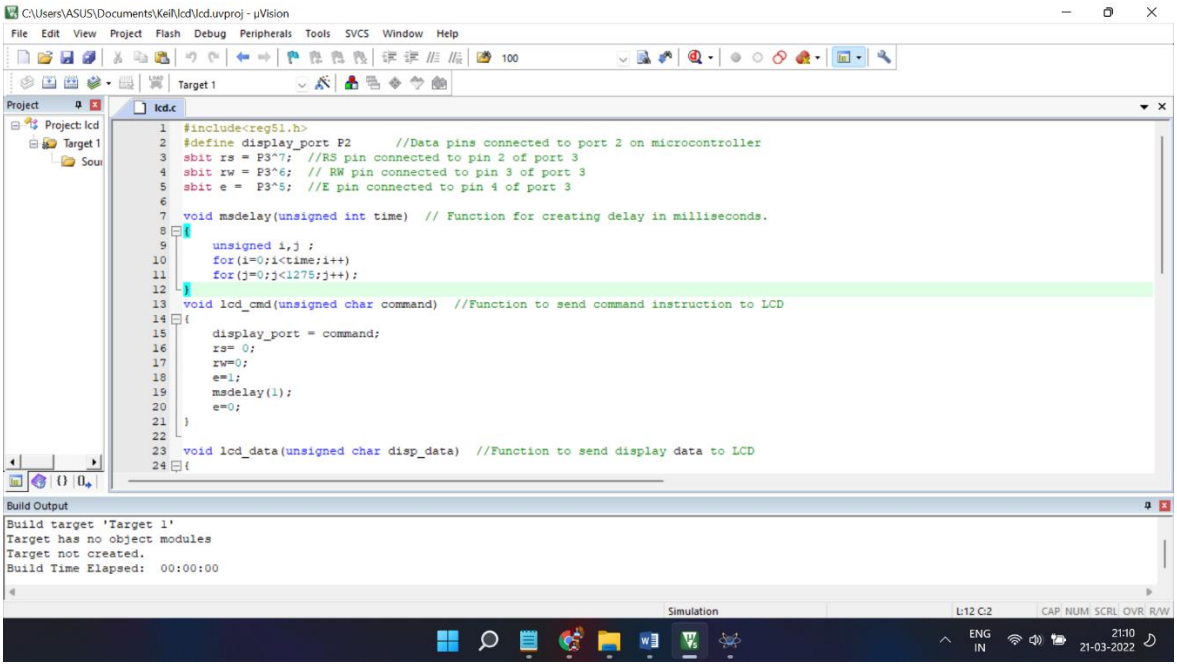
LCDs cannot form multiple resolution images.
The contrast ratio for LCD images is lesser than CRT and plasma displays.
Due to their longer response time, LCDs show ghost images and mixing when images change rapidly. The narrow viewing angle of an LCD weakens the image quality in wider viewing angles.

Algorithm:

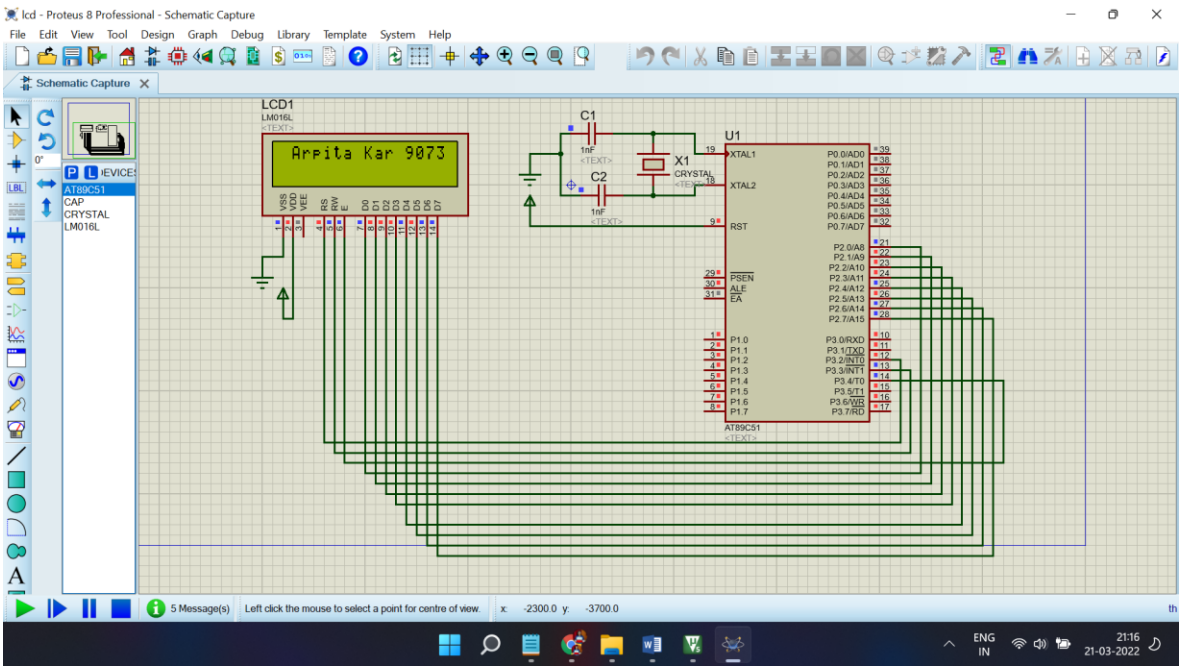
```
#include<reg51.h>
#define display_port P2      //Data pins connected to port 2 on microcontroller
sbit rs = P3^7;  //RS pin connected to pin 2 of port 3
sbit rw = P3^6;  // RW pin connected to pin 3 of port 3
sbit e =  P3^5;  //E pin connected to pin 4 of port 3
void msdelay(unsigned int time)  {
   unsigned i,j ;
   for(i=0;i<time;i++)
   for(j=0;j<1275;j++);
}
void lcd_cmd(unsigned char command)  //Function to send command instruction to LCD
{
   display_port = command;
   rs= 0;
   rw=0;
   e=1;
   msdelay(1);
   e=0;
}
void lcd_data(unsigned char disp_data)  {
   display_port = disp_data;
   rs= 1;
   rw=0;
```

```
    e=1;
    msdelay(1);
    e=0;
}
 void lcd_init()    //Function to prepare the LCD  and get it ready
{
    lcd_cmd(0x38);  // for using 2 lines and 5X7 matrix of LCD
    msdelay(10);
    lcd_cmd(0x0F);  // turn display ON, cursor blinking
    msdelay(10);
    lcd_cmd(0x01);  //clear screen
    msdelay(10);
    lcd_cmd(0x81);  // bring cursor to position 1 of line 1
    msdelay(10);
}
void main() {
    unsigned char a[100]="Arpita Kar 9073.";    //string of 11 characters with a null terminator.
    int l=0;
    lcd_init();
    while(a[l] != '\0') // searching the null terminator in the sentence
    {
        lcd_data(a[l]);
        l++;
        msdelay(10);
    }
     msdelay(100);
}
```

Keil Screenshot:



Proteus Screenshot:



Post- Lab Questions

1. Explain the interfacing signals between the 8051 and the LCD controller.

2. Draw and explain the interface of the touch screen controller to the 8051.

3. Explain the functions of the initialization sequence of the LCD.

PAGE No. ①
DATE / /

EXP - 1

Q.1] The interfacing signals between 8051 & LCD are as follows:

① Vcc, Vss & VEE:
Vcc provides +5V power while Vss provides ground connection. VEE is used to control the contrast of the LCD.

② RS (Register Select):
If RS = 0, the data instruction command code register is selected allowing users to send a command such as clear display, etc.
If RS = 1, the data registered is selected allowing user to send data to be displayed on the LCD.

③ R/W (Read/Write):
It allows user to write information on LCD or read information from it. Here, R/W = 0 for sending & R/W = 1 for receiving.

④ E (Enable):
This pin is used by LCD to latch information presented to it by the data pins.

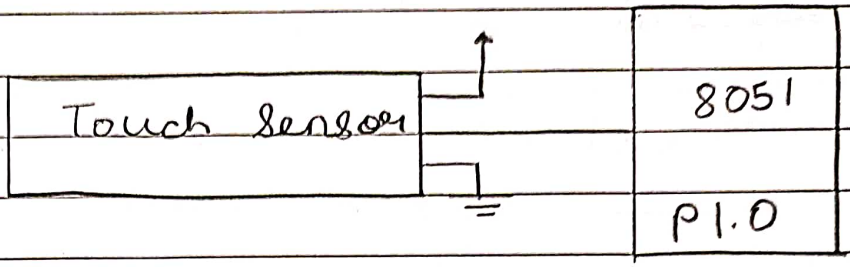
⑤ D0 - D7 (Data lines):
The 8-bit data pins are used to send information to the LCD or to read contents off an internal LCD register. In order to display letters, characters or numbers, we

9073

PAGE No. 2
DATE / /

EXP - 1

Q 2.]



> Touch sensors are simple transducers whereas touch screens has a resistive layer in both X & Y direction.
> According to position of touch, their X & Y channel resistance change.
> So we have to determine the X & Y channel's resistance to get the position of touch in terms of X & Y coordination.
> We would use an ADC to convert the analog co-ordinates from the sensor to digital values for the 8051 microcontroller.

Q.3.] > LCD 16 x 2 can be used in 4-bit mode or 8-bit mode depending on the application. In order to use it, we need to send certain commands to the LCD in command mode and once the LCD is configured to our need, we can send the required data in data mode.
> The steps for initializing the LCD display is follows and is common for almost all applications.

PAGE No. 3
DATE / /

EXP - 1

1. Send 38H to the 8-bit data line for initialization.
2. Send 0FH for making the LCD 'ON', cursor 'ON' and cursor blinking 'ON'
3. Send 06H for incrementing cursor position.
4. Send 01H for clearing the display and return the cursor

> These commands can also be setup to run as an initialization code when the LCD first boots up when connected to 8051.

eg;

```
Void LCD - Init (void)
{
 delay (20);
 LCD - Command (0x38);
 LCD - Command (0x0C);
 LCD - Command (0x06);
 LCD - Command (0x01);
 LCD - Command (0x80);
```

Fr. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING (FrCRCE)
Department of Electronics and Computer Science (ECS)

2. STEPPER MOTOR INTERFACING

Course, Subject & Experiment Details

Academic year	**2021 – 2022**	**Estimated Time**	**02 Hours**
Course	**T.E. (ECS)**	**Subject Name**	**Embedded systems and RTOS**
Semester	**VI**	**Chapter Title**	**Motor interfacing**
Experiment Type	**Coding**	**Subject Code**	**ECC 601**

Aim & Objective of Experiment

1. To write and execute an assembly language program to interface a stepper motor to the 8051 (Using KEIL Micro-Vision) with direction control

2. To design the system (using Proteus VSM) and show simulation results.

Theory:

Stepper motors rotate or step from one fixed position to the next in small increments. Common step size range from 0.9° to 30°.Stepper motors are stepped from one position to the next by changing the currents to the field of the motor. The two common field connections are required to as two phases and four phases. Stepper motor control involves interfacing the motor coil connections to the microcontroller port via a driver circuit that provides the necessary drive current for the motor. The microcontroller generates patterns on the port providing the excitation to the field coils which cause it to rotate. The tables below show the switching sequence for the typical stepper motor for full step mode and half step mode. In full step mode motor, steps through the step angle each time it is excited. In half step mode, motor steps through half the mode angle each time. This is accomplished by maintaining one field excitation at a time and changing the other field. Each the microcontroller output one step code it must wait a few milliseconds before the next step code is given out because the motor can only step so fast.

Full Step Drive Sequence:

	D	C	B	A
1	**0**	**0**	**1**	**1**
2	**1**	**0**	**0**	**1**
3	**1**	**1**	**0**	**0**
4	**0**	**1**	**1**	**0**

Half Step Drive Sequence:

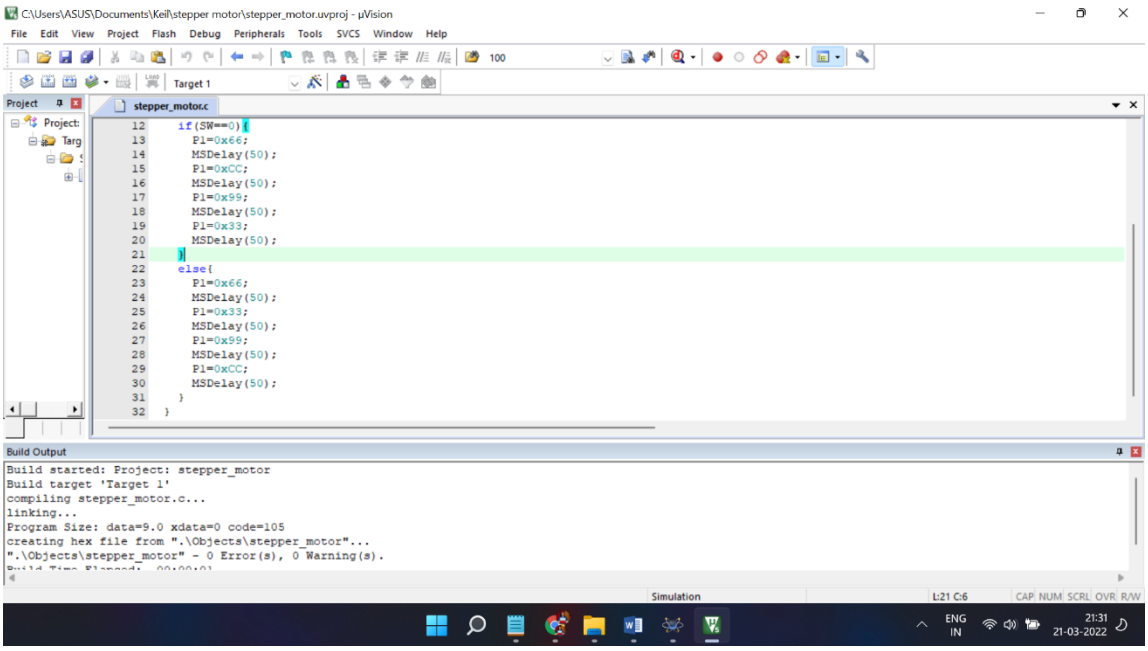
	D	C	B	A
1	**0**	**0**	**1**	**1**
2	**0**	**0**	**0**	**1**
3	**1**	**0**	**0**	**1**
4	**1**	**0**	**0**	**0**
5	**1**	**1**	**0**	**0**
6	**0**	**1**	**0**	**0**
7	**0**	**1**	**1**	**0**
8	**0**	**0**	**1**	**0**

Algorithm:

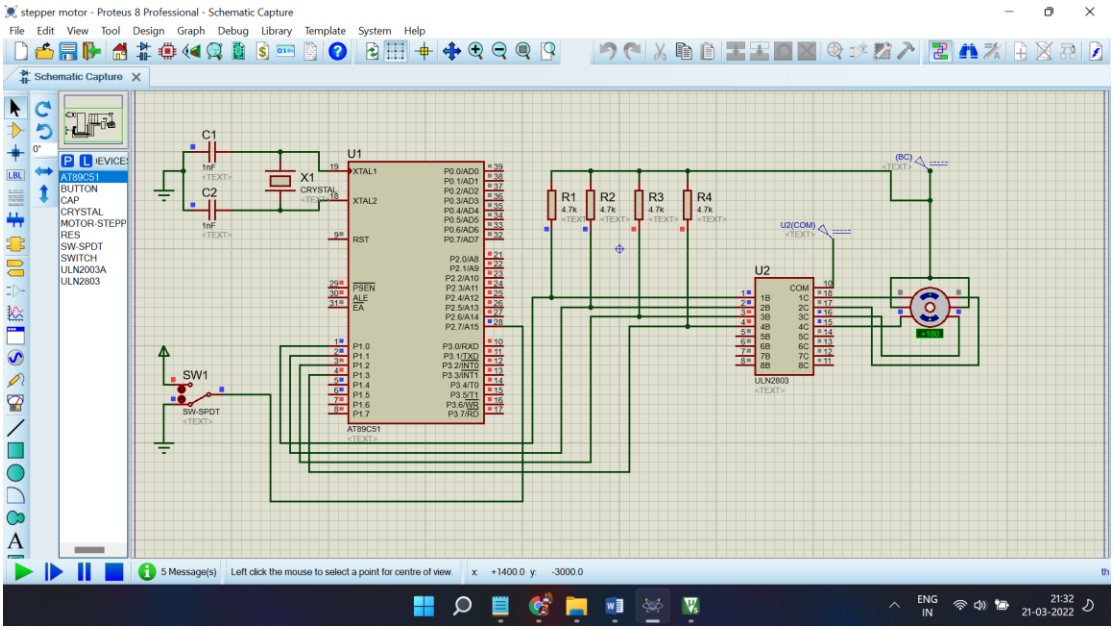
```
#include <reg51.h>
sbit SW=P2^7;
void MSDelay(unsigned int value){
   unsigned int x,y;
   for(x=0;x<1275;x++)
   for(y=0;y<value;y++);
}
void main(){
   SW=1;
   while(1){
    if(SW==0){
          P1=0x66;
          MSDelay(50);
          P1=0xCC;
          MSDelay(50);
          P1=0x99;
```

```
            MSDelay(50);
            P1=0x33;
            MSDelay(50);
        }
        else {
            P1=0x66;
            MSDelay(50);
            P1=0x33;
            MSDelay(50);
            P1=0x99;
            MSDelay(50);
            P1=0xCC;
            MSDelay(50);
        }
      }
    }
```

Keil Screenshots:



Proteus Screenshots:



Post- Lab Questions

1. Study the hardware setup of the motor. Explain why the clamp diodes are used across the driving transistors.
2. Why is a slight jerking motion induced in the motor motion in the full step drive?

9073

PAGE No. ① DATE / /

Exp-2

Q. 1.]

a) ① The hardware setup of a stepper motor is fairly related to a DC motor. It includes a permanent magnet like Rotor which is in the middle & It will turn once force acts on it. This rotor is enclosed through a no. of stator which is wound through a magnetic coil all over it. The stator is arranged near the rotor so that magnetic fields within the stator can control the movement of the rotor.

② The stepper motor can be controlled by energizing every stator one by one. The stator would the magnetize and work like an electromagnetic pole which uses repulsive energy on the rotor to move forward. The stator's alternative magnetizing & de-magnetizing will shift the rotor gradually & allow it to turn through great control.

b) Diodes are used to reduce the back EMF spike created when the coils are energized & de-energized, similar to how electro-mechanical relays function. TIP transistors can be used to supply higher current to the motor as they can accomodate higher voltages & power draws.

9073

PAGE No. ②
DATE / /

EXP-2

Q.2.]

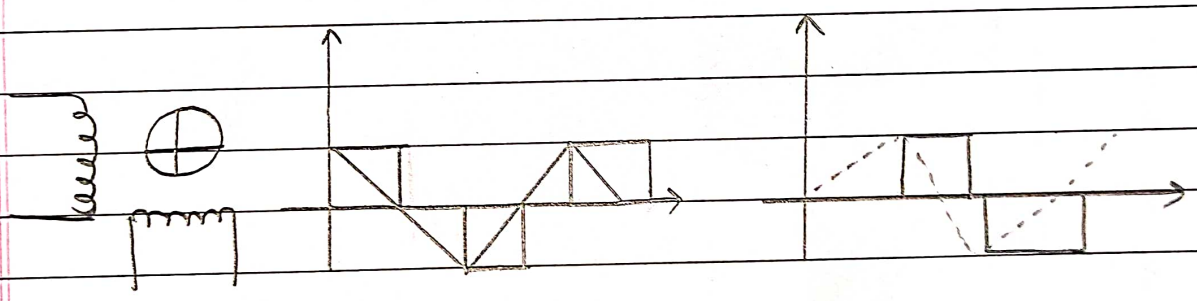
① During a full-set drive, the stepper motor driver energizes two coils of the stepper motor in a pulse / direction command.

② Each pulse of drive mode causes motor to move on basic step angle.

③ The current vector drawn by full-step divides a circle into 4 equal-parts and current waveform is rough.

④ With this, the motor would vibrate at lower speeds and noise will be generated.

⑤ Below is the current vector segment ion diagram and IVST diagram.



3. Sensor Interfacing

Course, Subject & Experiment Details

Academic year	**2021 – 2022**	**Estimated Time**	**02 Hours**
Course	**T.E. (ECS)**	**Subject Name**	**Embedded systems and RTOS**
Semester	**VI**	**Chapter Title**	**Sensor interfacing**
Experiment Type	**Coding**	**Subject Code**	**ECC 601**

Aim & Objective of Experiment

1. To write and execute an assembly language program to interface a Temperature sensor using an ADC to the 8051 (Using KEIL Micro-Vision)

2. To design the system (using Proteus VSM) and show simulation results.

Theory:

Analog to digital converters are among the most widely used devices for data acquisition. digital computers use binary (discrete) values but in the physical world, all parameters are analog (continuous) in nature. Therefore, we need an analog to digital converter to translate the analog signals to digital to be processed by the microcontroller.

An ADC has n bit resolution where n can be 8,10, 12,16 or 24 bits. The higher resolution ADC provides a smaller step size where step size is the smallest change that can be discerned by an ADC. In addition to resolution, conversion time is another major factor in choosing an ADC. Conversion time is defined as the time it takes for the ADC to convert from analog to digital. ADC chips can be serial or parallel. In parallel ADCs, multiple pins are used as output but in serial ADC, we have only one pin for data out.

Note : ADC 0808/0809 is a 8 bit ADC chip from National Semiconductors. It is a 8 channel ADC and each channel can be selected by using three select lines.

LM 35:The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in ˚ Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±1⁄4˚C at room temperature and ±3⁄4˚C over a full −55 to +150˚C temperature range.

Algorithm:

```
#include<reg51.h>

#include<string.h>


sbit RS = P2^5;

sbit RW = P2^6;

sbit EN = P2^7;

sbit ale=P2^3;

sbit oe=P2^4;

sbit start=P2^1;

sbit eoc=P2^0;

sbit clk=P2^2;

sbit chc=P0^7;  //Address pins for selecting input channels.

sbit chb=P0^6;

sbit cha=P0^5;


void delay(int t);

void lcd_init(void);

void lcd_command(char c);

void lcd_data(char d);

void str(char a[]);

void print( long float p);
```

```
long float k;
unsigned long int q,r,x,y,z;


void timer0() interrupt 1  // TIMER 0 interrupt ISR
{
   clk=~clk;
}


void main()          // MAIN PROGRAM
{
lcd_init();                          // lcd initialization
str("!!welcome!!");
lcd_command(0x01);  // clear display
str("Temp:");
lcd_command(96); //custom character (°c) display
lcd_data(0x10);
lcd_data(0x07);
lcd_data(0x08);
lcd_data(0x08);
lcd_data(0x08);
lcd_data(0x08);
lcd_data(0x07);
lcd_command(0x8b);
lcd_data(4);
eoc=1;           // make eoc an input
ale=0;
oe=0;
```

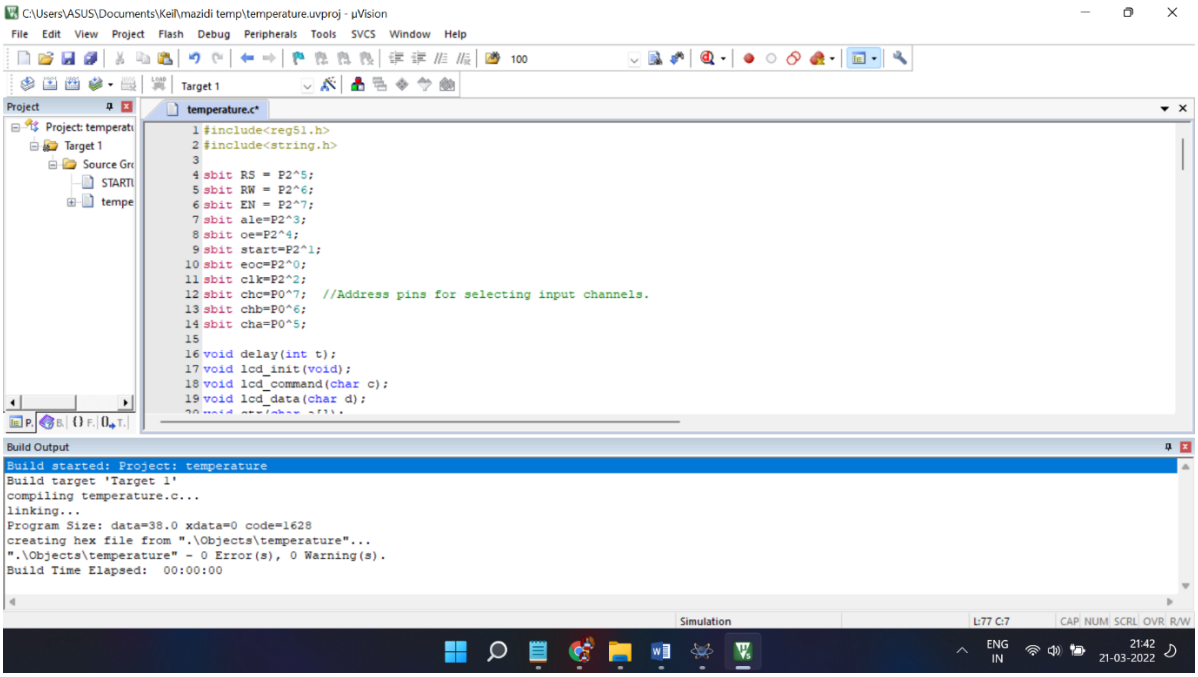
```
start=0;
TMOD=0x02;    // timer 0 in mode 2
TH0=0xc2;                //     15khz
IE=0x82; //  set timer 0 interrupt
TR0=1;   // start timer 0
while(1)
{
chc=0;  // select channel 0
chb=0;
cha=0;
ale=1;  // send high to low pulse on start and ale pin
start=1;
delay(1);
ale=0;
start=0;
while(eoc==1);     // wait for conversion
while(eoc==0);
oe=1;
k=P1;
lcd_command(0x85);
print(k);   // send the digital data to lcd
oe=0;
}
}

void str(char a[])  // lcd function to display string
{
```

```
int j;
for(j=0;a[j]!='\0';j++)
{
lcd_data(a[j]);
}
}

void lcd_init(void)  // lcd initialization
{
lcd_command(0x38);        //8 bit,2 line,5x8 dots
lcd_command(0x01);        // clear display
lcd_command(0x0f);      // display on, cursor blinking
lcd_command(0x06);     //Entry mode
lcd_command(0x0c); //cursor off
lcd_command(0x80); //// force cursor to beginning of first row
}

void lcd_command(char c) // lcd command function
{
P3=c;
RS=0;
RW=0;    // select command register
EN=1;
delay(5);
EN=0;
delay(5);
}
```

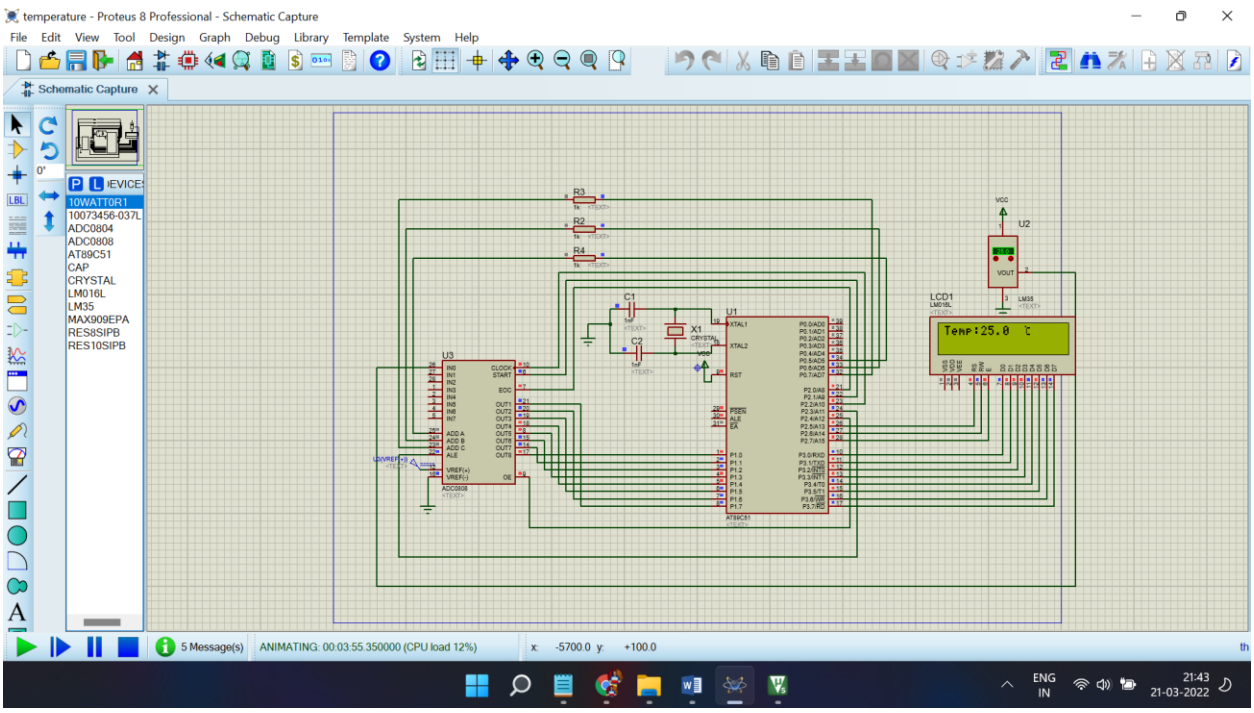
```
void lcd_data(char d) // lcd data function
{
P3=d;
RS=1;             //select data register
RW=0;
EN=1;
delay(5);
EN=0;
delay(5);
}

void delay(int t)      // delay function
{
int j;
for(j=0;j<t*1275;j++);
}

void print( long float p)  // number display function
{
x=p*10;
if(x>=1000)
{
q=x/1000;
q=q+48;
y=(x%1000)/100;
y=y+48;
```

```
z=((x%1000)%100)/10;
z=z+48;
r=x%10;
r=r+48;
lcd_data(q);
lcd_data(y);
lcd_data(z);
lcd_data(46);      //ascii value of point
lcd_data(r);
}

else
{
q=x/100;
q=q+48;
y=(x%100)/10;
y=y+48;
z=x%10;
z=z+48;
lcd_data(q);
lcd_data(y);
lcd_data(46);                  //ascii value of point
lcd_data(z);
r=0;
lcd_data(r);
}
}
```

Keil Screenshot:



Proteus Screenshot:



Post- Lab Questions

1. Explain the significance of the Vref terminal in the ADC 0808.

2. Explain the handshaking sequence with a neat timing diagram.

3.Explain the successive approximation technique for A to D conversion.

9073

PAGE No. ① DATE / /

EXP-3

Q 1.]

① These are two pins namely Vref(+) & Vref(-) in ADC 0808.

② These two pins are responsible to provide the upper and lower limit of voltages.

③ Vref voltages further help in determing the step range / size for the conversions.

④ Vref(+) would have a higher voltage & Vref(-) has the lower voltage

⑤ If Vref(+) has an input voltage 5V and Vref(-) has voltage of 0V, Then the step size would be $\frac{5V - 0V}{2^8} = 15.53\,mV$

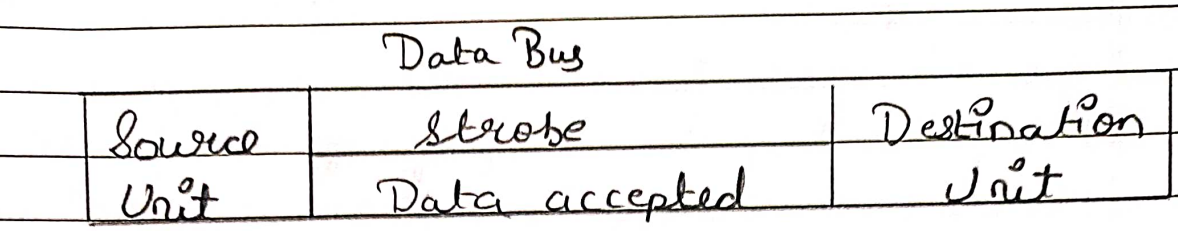
Q.2.] > Hand shacking is an I/O control approach in order to synchronize external devices with the microprocessor, as several devices send or receive data at a much lower rate than the microprocessor.

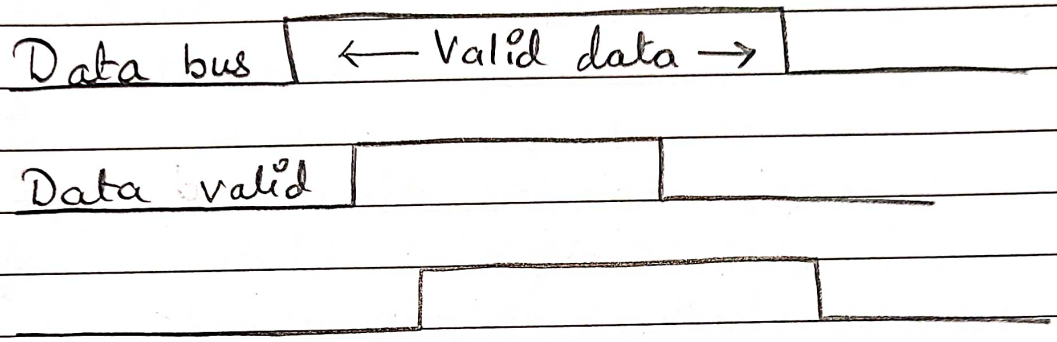
> In the hand shacking process, first a signal is sent from the source channel to the destination. Then the destination sends data and acknowledges by sending a signal informing which destination channels are available to receive further data.

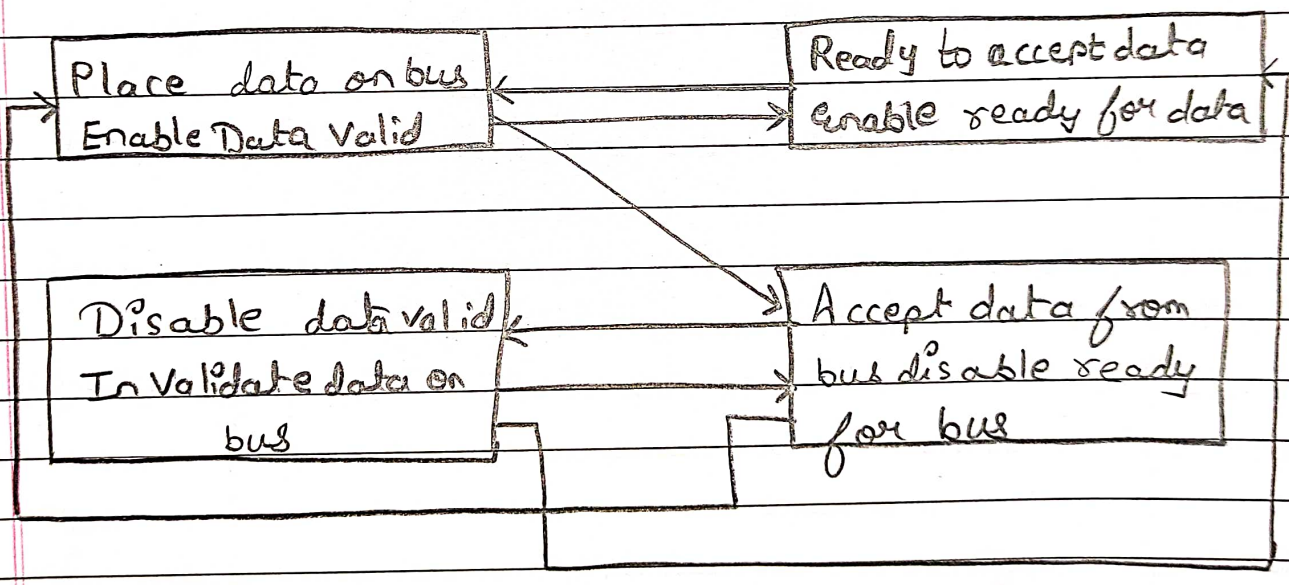
EXP-3

9073

PAGE No. 2
DATE / /







> The diagram shows the data transfer process when initiated by the source, the two handshacking lines are data valid, which is created by the source unit and data accepted created by the destination unit. The timing diagram displays the exchange of signals between two units.

> The sequence of events shows the four possible states that the system can be

at any given time. The source unit initiates the transfer by locating the information on the bus and enabling it's data valid signal. The data accepted signal is activated by the destination until after it obtains the data from the bus.

> The name of the signal created by the destination unit has been modified to ready for the data to reflect its new destination. The source unit in this case does not locate data on the bus until it takes the ready for data signal from the destination unit.

> The sequence of cases in both events would be equal if it can consider the ready for data signal as complement of data accepted. The only difference initial transfer is in their choice of the initial state.

Q.3.]

① Successive approximation type ADC is the most widely used & popular method. The conversion time is maintained constant in successive approximation type ADC and is proportional to the number of bits in the digital output.

② The basic principle of this type of ADC is that the unknown analog input voltage is approximated against an n-bit digital value by trying 1 bit at a time

beginning with the MSB.

(3) The principle of successive approximation process for a 4 bit conversion is as follows this type of ADC operates by successively dividing the voltage range by half.

S·1: The MSB is initially set to 1 while the other bits are set to 000. The digital equivalent voltages is compared with the unknown analog input voltage.

S·2: If the analog input voltage is higher than the digital equivalent voltage, the MSB is retained as 1 and another 2nd MSB is set to 1, otherwise the 1st MSB is set to 0 and 2nd MSB is set to 1. The comparision is made in order to determine whether to retain or visit the 2nd MSB.

Eg.: Assume that a 4-bit ADC is used & the analog input voltage is $V_A = 11V$, when the conversion starts, the MSB is set to 1.

- Now
  $V_A = 11\,V > V_D = 8\,V = [1000]^2$
- Since the unknown analog input voltage V_A is higher than the equivalent digital voltage V_D, the MSB is retained as 1 and the next MSB is set to 1.
  $\therefore V_D = 12\,V = [1100]^2$
- Now
  $V_A = 11V < V_D = 12\,V = [1100]^2$
- The unknown analog input voltage V_A is lower than the equivalent digital voltage

EXP-3

V_D. Hence the 2nd MSB is set to 0 and next MSD is set to 1.

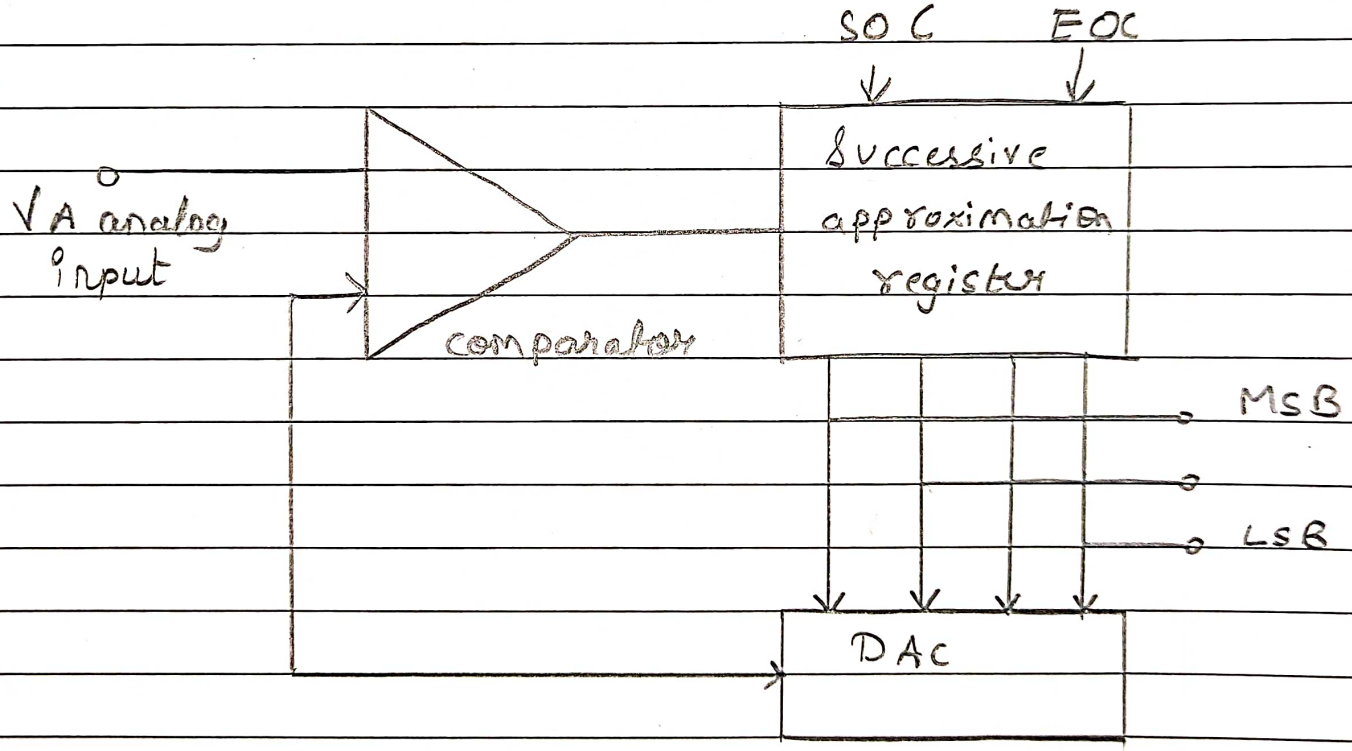
$\therefore V_D = 10V = [1010]^2$

Again,

$V_A = 11V > V_D = 10V = [1010]^2$

As $V_A > V_D$, the 3rd MSB is retained at 1 & last bit is set to 1.

$\therefore V_D = 11V = [1011]^2 \Rightarrow V_A = V_D$. Hence the conversion stops.



This block diagram consists of a successive approximation register (DAC) and a comparator. The output of SAR is given to n-bit DAC. The equivalent analog output voltage (V_D) of DAC, is applied to the non-inverting input of the comparator. The second input to the comparator is used to activate

9073

EXP-3

PAGE No. 6
DATE / /

the submissive approximation layer of SAR.

- When start command is applied, the SAR sets the MSB to logic 1 and other bits and mode to logic 0, so that trial codes becomes 1000.

4. DC motor Interfacing

Course, Subject & Experiment Details

Academic year	**2021 – 2022**	**Estimated Time**	**02 Hours**
Course	**T.E. (ECS)**	**Subject Name**	**Embedded systems and RTOS**
Semester	**VI**	**Chapter Title**	**Motor interfacing**
Experiment Type	**Coding**	**Subject Code**	**ECC 601**

Aim & Objective of Experiment

To drive a DC Motor and control its direction and speed.

Theory:

DC Motor ideally needs a constant voltage to rotate. Thus, to achieve speed control, signal conditioning is necessary. To generate a variable dc voltage using microcontroller either DAC is used or in some advanced application PWM is used. In PWM, control average DC voltage is varied by varying the ratio of ON time to OFF time.

$$VDC \propto Ton / Ton + Toff$$

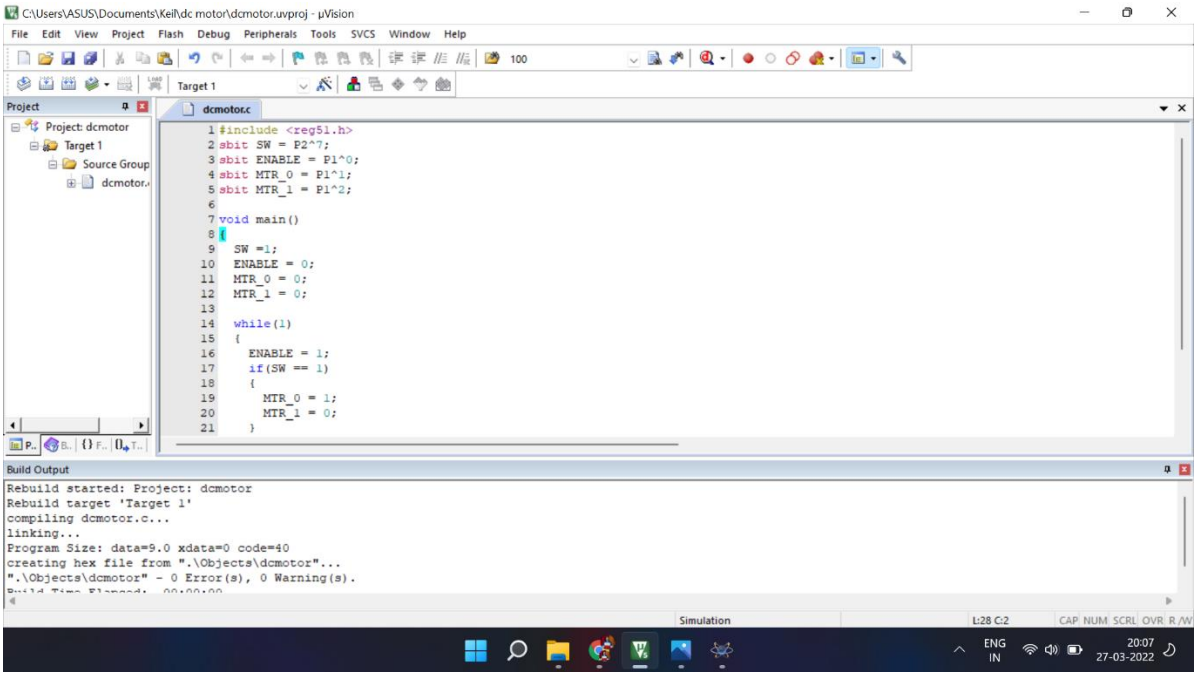
With the help of relays or some specially designed chips we can change the direction of the motor rotation.

H- Bridge control can be created using relays, transistors, or a single IC solution such as the L293. When using relays and transistors, you must ensure those invalid configurations do not occur. Be aware that the L293 will be generating heat during operation. For sustained operation of the motor, use a heat sink.

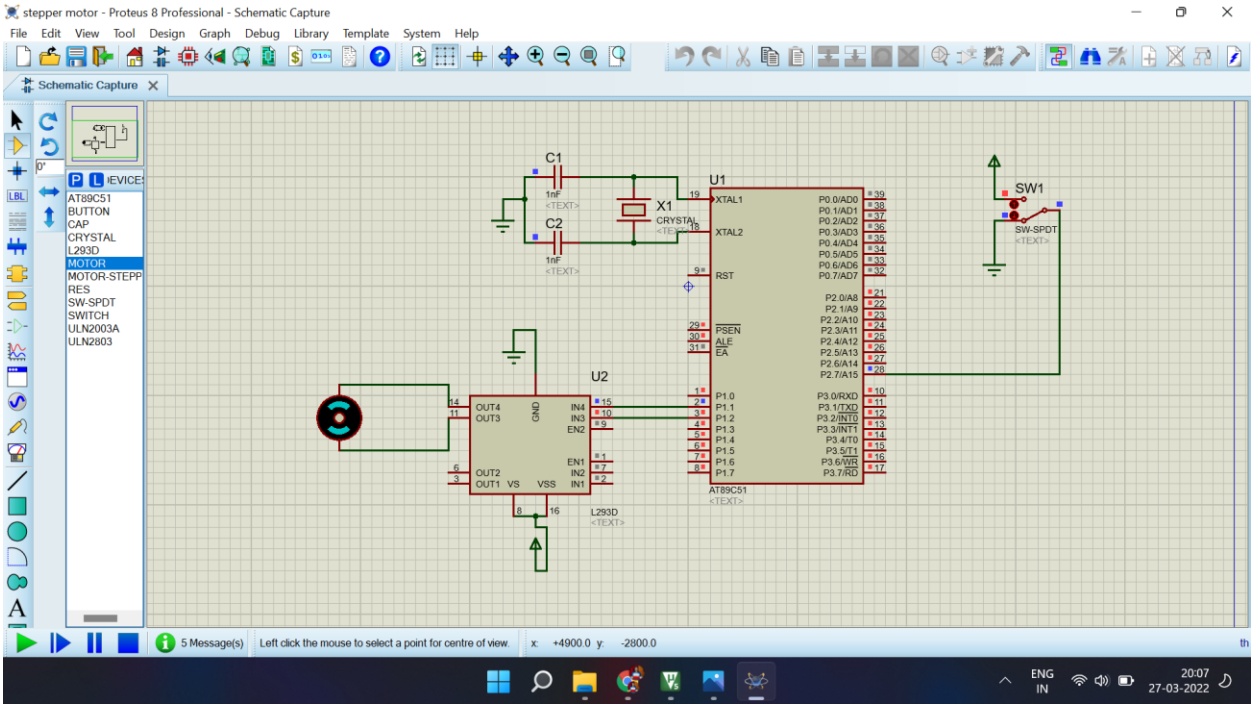
Algorithm:

```
#include <reg51.h>
sbit SW = P2^7;
sbit ENABLE = P1^0;
sbit MTR_0 = P1^1;
sbit MTR_1 = P1^2;
void main() {
        SW =1;
        ENABLE = 0;
        MTR_0 = 0;
        MTR_1 = 0;
        while(1)
        {
            ENABLE = 1;
            if(SW == 1) {
                MTR_0 = 1;
                MTR_1 = 0;
            }
            else {
                MTR_0 = 0;
                MTR_1 = 1;
            }
        }
}
```

Keil Screenshot:



Proteus Screenshot:



Post- Lab Questions

1. Give sample specifications/ratings of any DC motor

2. Explain PWM control of DC motor

9073

EXP-4

PAGE No. 1
DATE / /

Q.1.] DC brushless motor ratings:

1. Manufacturer = Seimens
2. Rated voltage = 932 V
   Maximum voltage = 1603 V
3. Rated current = 200 A
   Maximum current = 277 A
4. Rated speed = 2000 RPM
   Maximum speed = 3562 RPM
5. Motor weight = 1120 Kg
6. Rated Torque = 200 N.m
7. Rated Power = 186.4 KW
8. Torque coefficient = 1 Nm/A
9. Resistance of phase = 4.9 Ω
10. Inductance of phase = 1.6 mH
11. Number of poles = 4

Q.2.]

1. For a given motor, the speed of that motor depends on three factors:
a) load; b) voltage; c) current. For a given fixed load we can maintain a steady speed by using a method called pulse width modulation.

2. By changing the width of the pulse applied to the DC motor, we can increase or decrease the amount of power applied to the motor, thereby controlling the speed.

EXP - 4

9073

PAGE No. (2)
DATE / /

(3) For microcontroller without the PWM circuitry, we must create the various duty cycles (pulses) using software, which prevents the microcontroller from doing other things.

(4) The ability to control the speed of the DC motor using PWM is one reason that DC motor are preferable over AC motors.

5. Arduino programming (Sensor Interfacing)

Course, Subject & Experiment Details

Academic year	**2021 – 2022**	**Estimated Time**	**02 Hours**
Course	**T.E. (ECS)**	**Subject Name**	**Embedded systems and RTOS**
Semester	**VI**	**Chapter Title**	**Arduino Programming**
Experiment Type	**Coding**	**Subject Code**	**ECC 601**

Aim & Objective of Experiment

To interface a Humidity-Temperature /Ultrasonic Sensor to the Arduino Uno (Proteus VSM /Hardware)

Theory:

Arduino UNO is a microcontroller board based on the **ATmega328P**. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. ATmega328P 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash.

The Arduino Integrated Development Environment - or **Arduino Software (IDE)** - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino hardware to upload programs and communicate with them.

Writing Sketches

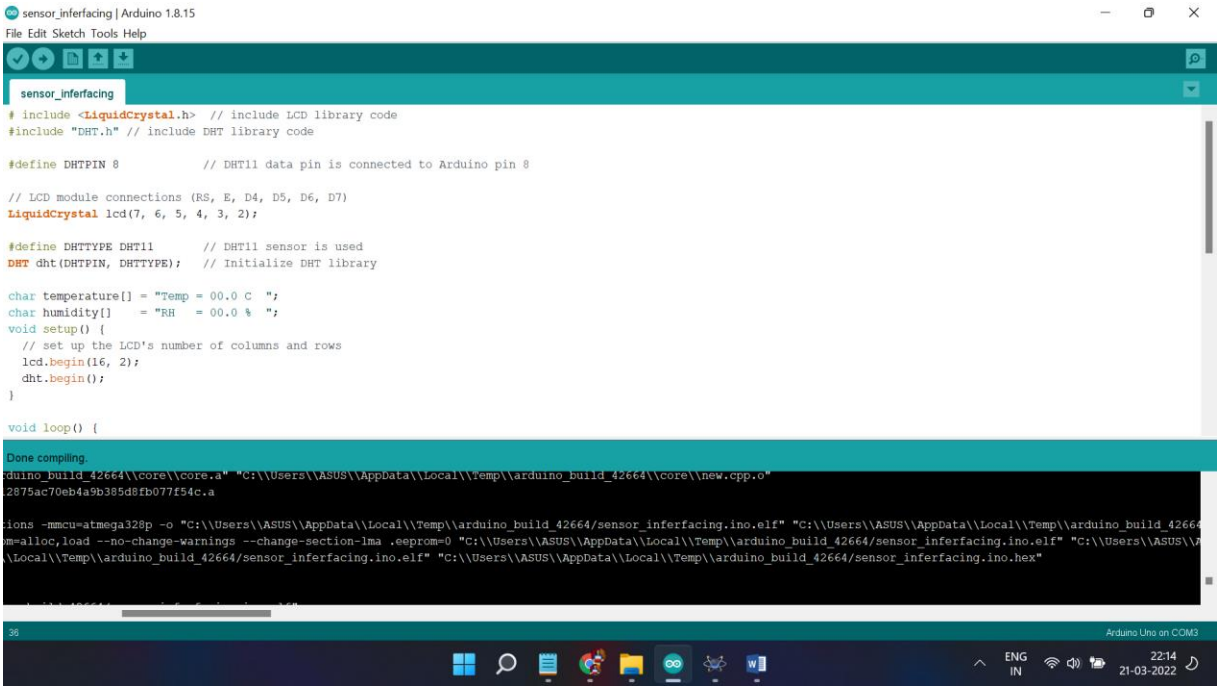
Programs written using Arduino Software (IDE) are called **sketches**. These sketches are written in the text editor and are saved with the file extension .ino. The editor has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom righthand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

Algorithm:

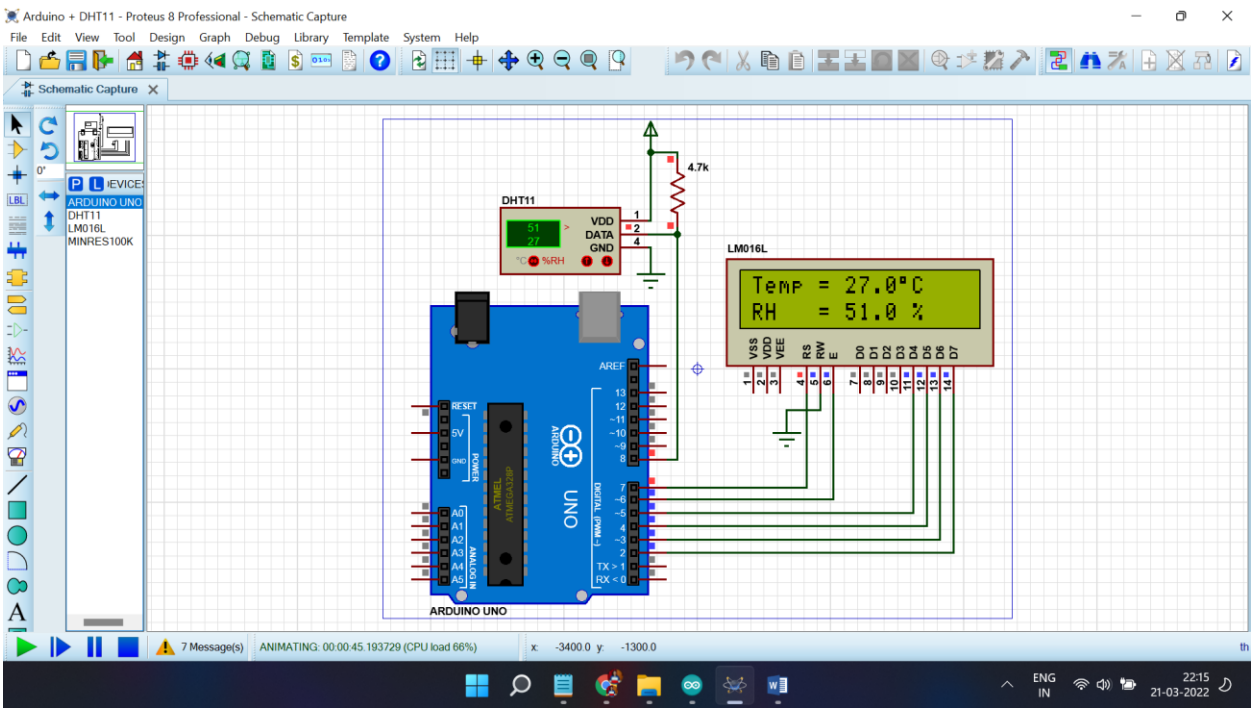
```
# include <LiquidCrystal.h>  // include LCD library code

#include "DHT.h" // include DHT library code


#define DHTPIN 8           // DHT11 data pin is connected to Arduino pin 8


// LCD module connections (RS, E, D4, D5, D6, D7)

LiquidCrystal lcd(7, 6, 5, 4, 3, 2);


#define DHTTYPE DHT11       // DHT11 sensor is used

DHT dht(DHTPIN, DHTTYPE);   // Initialize DHT library


char temperature[] = "Temp = 00.0 C  ";

char humidity[]    = "RH   = 00.0 %  ";

void setup() {

  // set up the LCD's number of columns and rows

  lcd.begin(16, 2);

  dht.begin();

}


void loop() {

  delay(1000);          // wait 1s between readings

  // Read humidity

  byte RH = dht.readHumidity();
```

```
  //Read temperature in degree Celsius
  byte Temp = dht.readTemperature();


  // Check if any reads failed and exit early (to try again)
  if (isnan(RH) || isnan(Temp)) {
    lcd.clear();
    lcd.setCursor(5, 0);
    lcd.print("Error");
    return;
  }
//conversion of ecimal values to ascii FOR Lcd
  temperature[7]     = Temp / 10 + 48;
  temperature[8]     = Temp % 10 + 48;
  temperature[11]    = 223;
  humidity[7]        = RH / 10 + 48;
  humidity[8]        = RH % 10 + 48;
  lcd.setCursor(0, 0);
  lcd.print(temperature);
  lcd.setCursor(0, 1);
  lcd.print(humidity);
}
```

Arduino IDLE Screenshot:



Proteus Screenshot:



Post- Lab Questions:

1. Explain the principle of working of the sensor used
2. Explain all the Arduino functions used in the program

EXP-5

9073

PAGE No. 1
DATE / /

Q.1] DHT 11?

① A DHT 11 sensor consists of a capacitive humidity sensor element and a thermistor for sensing temperatures. The humidity sensing capacitor has two electrodes with a moisture holding substrate as a dielectric between them.

② Change in the capacitance value occurs with the change in humidity levels. The sensor measures processes these changed resistance values & changes them into digital form.

③ For measuring temperature, this sensor uses a negative temperature coefficient thermistor, which causes a decrease in its resistance value with increase in temperature. To get layer resistance values even for the smallest change in temperature, this sensor is usually made up of semiconductor ceramics or polymers.

④ The temperature range of DHT 11 is from 0 to 50°C with a ±2°C accuracy. Humidity range of this sensor is from 20% to 80% with a ±5% accuracy. The sampling rate of this sensor is 1Hz, i.e one reading for every second. DHT 11 is small in size with an operating voltage from 3 to 5 volts with max. current draw being 2.5mA.

EXP-5 9073 PAGE No. 2 DATE / /

⑤ DHT 11 sensor has four pins - Vcc, GND, Data and a NC pin. A pull-up resistor of 5k to 10k ohms is required for communication between the sensor & micro-controller.

Q.2: ① dht.begin(): This function is used to enable the library for the DHT sensor. It is used applied in the void setup section.

② Serial.begin(9600): This function is used to setup and begin the serial monitor for monitoring the outputs for the sensor. We have set a baud rate of 9600.

③ Serial.println(): This function is used to print outputs or sentences in the serial monitor.

④ dht.readHumidity(): This function extracts the humidity value from the sensor and allots a value to any assigned float variable.

⑤ dht.readTemperature(): This function extracts the temperature value from the sensor and allots a value to any assigned float variable.

⑥ isnan(): This function basically checks if these are values present in the provided variable or not.

9073

EXP-5

PAGE No. 3
DATE / /

(7) dht.computeHeatIndex() : This function calculate 'Heat-Index' provided we have given temperature & humidity.

6. Arduino programming (RTC Interfacing)

Course, Subject & Experiment Details

Academic year	**2021 – 2022**	**Estimated Time**	**02 Hours**
Course	**T.E. (ECS)**	**Subject Name**	**Embedded systems and RTOS**
Semester	**VI**	**Chapter Title**	**Arduino Programming**
Experiment Type	**Coding**	**Subject Code**	**ECC 601**

Aim & Objective of Experiment

Interfacing Real Time Clock (DS1307) with Arduino Controller

Theory:

Real Time Clock or RTC is a system that keeps track of the current time and can be used in any device which needs to keep accurate time. RTCs often have an alternate source of power, so that they can continue to keep time while the primary source of power is off or unavailable. You can also keep tracking the exact time without using RTC systems, but RTCs have some important advantages. Here are some of these advantages:

- Low power consumption
- Releasing system time from time calculation (this feature is critical because in many cases CPU is operating some delicate tasks like receiving sensors data. and if you don't use RTC, CPU also has to keep track of the time and it can disrupt processors main tasks.)
- High accuracy

DS1307 Module Feature & Specifications

DS1307 module is one of the most affordable and common RTCs modules. It can accurately keep track of seconds, minutes, hours, days, months, and years.

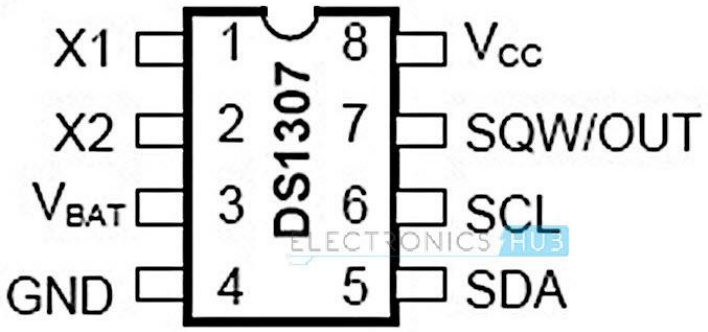
Some of the DS1307 important features are:

- Ability of Generating Programmable Square-Wave

- Low Current Use; under 500nA in Battery Backup mode
- The Ability to Set the Date Up to Year 2100
- I2C Serial Interface

DS1307 RTC Pin Diagram

The following image shows the pin diagram of the DS1307 RTC IC. In order to reduce the power consumption, the number of pins on the IC has to be reduced. Hence, DS1307 RTC used I2C Communication.



Algorithm:

```
#include <LiquidCrystal.h>
// include Wire library code (needed for I2C protocol devices)
#include <Wire.h>

// LCD module connections (RS, E, D4, D5, D6, D7)
LiquidCrystal lcd(2, 3, 4, 5, 6, 7);

void setup() {
  pinMode(8, INPUT_PULLUP);                  // button1 is connected to pin 8
  pinMode(9, INPUT_PULLUP);                  // button2 is connected to pin 9
  // set up the LCD's number of columns and rows
  lcd.begin(16, 2);
  Wire.begin();                              // Join i2c bus
}

char Time[]     = "TIME:  :  :  ";
char Calendar[] = "DATE:  /  /20  ";
byte i, second, minute, hour, date, month, year;

void DS3231_display(){
  // Convert BCD to decimal
  second = (second >> 4) * 10 + (second & 0x0F);
  minute = (minute >> 4) * 10 + (minute & 0x0F);
  hour   = (hour >> 4)   * 10 + (hour & 0x0F);
  date   = (date >> 4)   * 10 + (date & 0x0F);
  month  = (month >> 4)  * 10 + (month & 0x0F);
```

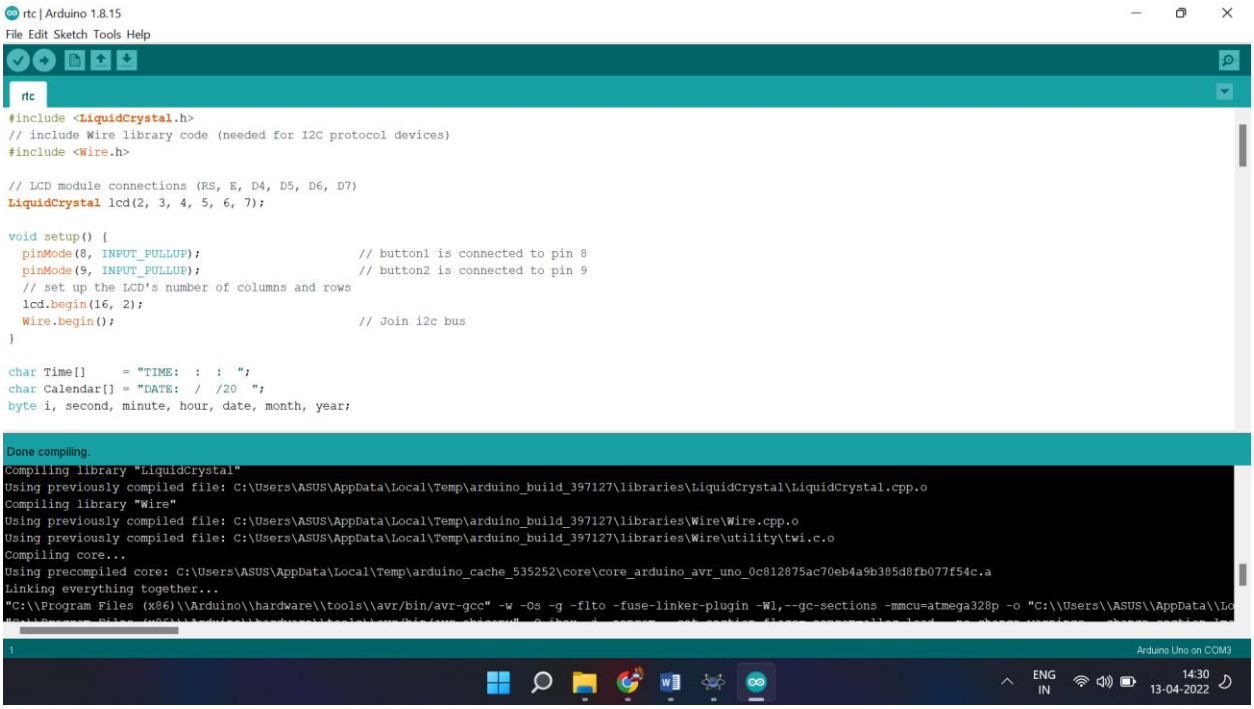
```
  year   = (year >> 4)   * 10 + (year & 0x0F);
  // End conversion
  Time[12]     = second % 10 + 48;
  Time[11]     = second / 10 + 48;
  Time[9]      = minute % 10 + 48;
  Time[8]      = minute / 10 + 48;
  Time[6]      = hour   % 10 + 48;
  Time[5]      = hour   / 10 + 48;
  Calendar[14] = year   % 10 + 48;
  Calendar[13] = year   / 10 + 48;
  Calendar[9]  = month  % 10 + 48;
  Calendar[8]  = month  / 10 + 48;
  Calendar[6]  = date   % 10 + 48;
  Calendar[5]  = date   / 10 + 48;
  lcd.setCursor(0, 0);
  lcd.print(Time);                      // Display time
  lcd.setCursor(0, 1);
  lcd.print(Calendar);                   // Display calendar
}
void blink_parameter(){
  byte j = 0;
  while(j < 10 && digitalRead(8) && digitalRead(9)){
    j++;
    delay(25);
  }
}
byte edit(byte x, byte y, byte parameter){
```

```
char text[3];
while(!digitalRead(8));                    // Wait until button (pin #8) released
while(true){
  while(!digitalRead(9)){                  // If button (pin #9) is pressed
    parameter++;
    if(i == 0 && parameter > 23)           // If hours > 23 ==> hours = 0
      parameter = 0;
    if(i == 1 && parameter > 59)           // If minutes > 59 ==> minutes = 0
      parameter = 0;
    if(i == 2 && parameter > 31)           // If date > 31 ==> date = 1
      parameter = 1;
    if(i == 3 && parameter > 12)           // If month > 12 ==> month = 1
      parameter = 1;
    if(i == 4 && parameter > 99)           // If year > 99 ==> year = 0
      parameter = 0;
    sprintf(text,"%02u", parameter);
    lcd.setCursor(x, y);
    lcd.print(text);
    delay(200);                            // Wait 200ms
  }
  lcd.setCursor(x, y);
  lcd.print("  ");                         // Display two spaces
  blink_parameter();
  sprintf(text,"%02u", parameter);
  lcd.setCursor(x, y);
  lcd.print(text);
  blink_parameter();
```

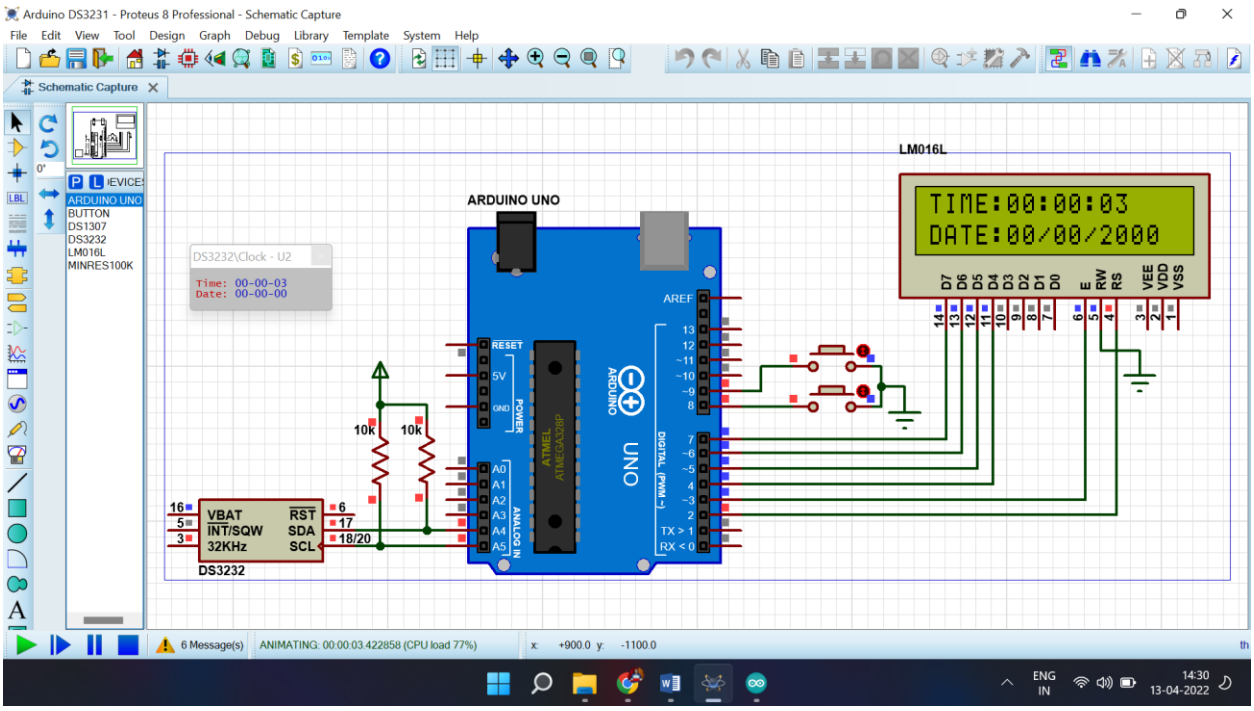
```
    if(!digitalRead(8)){                    // If button (pin #8) is pressed
      i++;                                  // Increament 'i' for the next parameter
      return parameter;                     // Return parameter value and exit
    }
  }
}

void loop() {
  if(!digitalRead(8)){                      // If button (pin #8) is pressed
      i = 0;
      hour   = edit(5, 0, hour);
      minute = edit(8, 0, minute);
      date   = edit(5, 1, date);
      month  = edit(8, 1, month);
      year   = edit(13, 1, year);
      // Convert decimal to BCD
      minute = ((minute / 10) << 4) + (minute % 10);
      hour = ((hour / 10) << 4) + (hour % 10);
      date = ((date / 10) << 4) + (date % 10);
      month = ((month / 10) << 4) + (month % 10);
      year = ((year / 10) << 4) + (year % 10);
      // End conversion
      // Write data to DS3231 RTC
      Wire.beginTransmission(0x68);             // Start I2C protocol with DS3231 address
      Wire.write(0);                            // Send register address
      Wire.write(0);                            // Reset sesonds and start oscillator
      Wire.write(minute);                       // Write minute
```

```
    Wire.write(hour);                   // Write hour
    Wire.write(1);                      // Write day (not used)
    Wire.write(date);                   // Write date
    Wire.write(month);                  // Write month
    Wire.write(year);                   // Write year
    Wire.endTransmission();             // Stop transmission and release the I2C bus
    delay(200);                         // Wait 200ms
  }
  Wire.beginTransmission(0x68);         // Start I2C protocol with DS3231 address
  Wire.write(0);                        // Send register address
  Wire.endTransmission(false);          // I2C restart
  Wire.requestFrom(0x68, 7);            // Request 7 bytes from DS3231 and release I2C bus at end of reading
  second = Wire.read();                 // Read seconds from register 0
  minute = Wire.read();                 // Read minuts from register 1
  hour   = Wire.read();                 // Read hour from register 2
  Wire.read();                          // Read day from register 3 (not used)
  date   = Wire.read();                 // Read date from register 4
  month  = Wire.read();                 // Read month from register 5
  year   = Wire.read();                 // Read year from register 6
  DS3231_display();                     // Diaplay time & calendar
  delay(50);                            // Wait 50ms
}
```

Arduino Code:



Output:



Post-Lab questions:

1. Why do RTCs normally use a 32.768 kHz crystal oscillator?
2. Explain the I2C communication used in the above experiment?

9073 Arpita Kar

PAGE No. 1
DATE / /

EXP-6

Q1.]

> The frequency of a real time clock varies with the application. The frequency 32768Hz is commonly used, because it is a power of 2 value (i.e., 2^{15}) and you get precisely 1sec period (1Hz frequency) by using a 15 stage binary counter.

> Practically, in majority of applications, particularly digital, the current consumption has to be as low as possible to preserve battery life. So, this frequency is elected as a best compromise between low frequency and convenient manufacture with market availability and estate in term of physical dimensions while designing board, where low frequency generally means the quartz is physically bigger.

Q.2.]

> The I2C-RTC board is a 6-pin CMOS real-Time clock Device using I2C-bus. There are no external components required. Only two signal lines SDA & SCL plus supply voltage and ground are required to be connected. This makes it perfect for embedded systems that require real time clock.

> The I2C protocol requires only 2 sig clock and data. Clock is known as SCL

9073 Arpita Kar

EXP-6

PAGE No. 12
DATE / /

or SCK while Dat is known as SDA. What makes I2C unique is the use of special combinations of signals conditions and changes. Fundamentally, there are just 2, start and stop. There are some other conditions which device from the presence or absence of these two, but they are the core of what make this bus unique. An I2C bus can potentially have multiple masters and many 'slave' device sharing the same bus, although you rarely see multiple masters in the real world. Signal contention is avoided by means of an open-collector drive scheme, where no device will ever drive a signal high, it only draws low, the bus is pulled up by a resistor.

Fr. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING (FrCRCE)

Department of Electronics and Computer Science (ECS)

7. Porting FreeRTOS on Arduino

Course, Subject & Experiment Details

Academic year	**2021 – 2022**	**Estimated Time**	**02 Hours**
Course	**T.E. (ECS)**	**Subject Name**	**Embedded systems and RTOS**
Semester	**VI**	**Chapter Title**	**FreeRTOS**
Experiment Type	**Coding**	**Subject Code**	**ECC 601**

Aim & Objective of Experiment

To port FreeRTOS an open source RTOS on the Arduino board

Theory:

FreeRTOS is an open source, real-time operating system for microcontrollers that makes small, low power edge devices easy to program, deploy, secure, connect, and manage. Distributed freely under the MIT open source license, FreeRTOS includes a kernel and a growing set of software libraries suitable for use across industry sectors and applications. This includes securely connecting your small, low power devices to AWS Cloud services like AWS IoT Core or to more powerful edge devices running AWS IoT Greengrass. FreeRTOS is built with an emphasis on reliability and ease of use, and offers the predictability of long-term support releases.

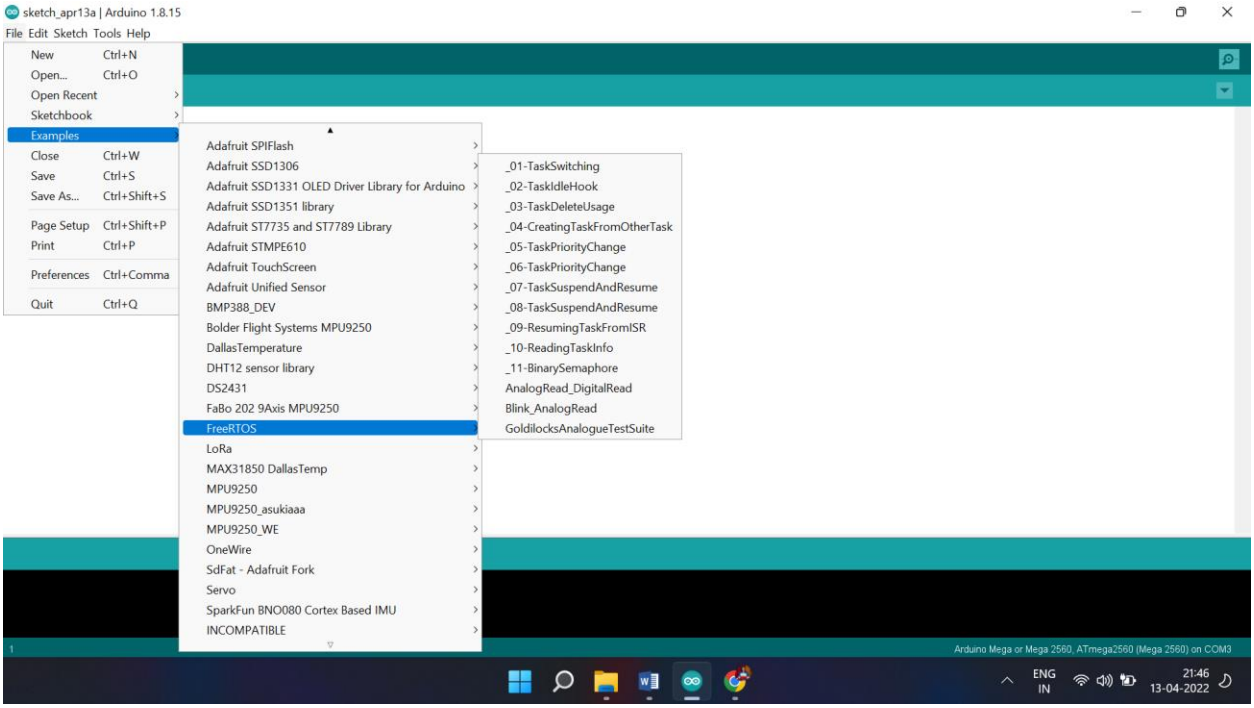
A microcontroller contains a simple, resource-constrained processor that can be found in many devices, including appliances, sensors, fitness trackers, industrial automation, and automobiles. Many of these small devices can benefit from connecting to the cloud or locally to other devices, but have limited compute power and memory capacity and typically perform simple, functional tasks. Microcontrollers frequently run operating systems that may not have built-in functionality to connect to local networks or the cloud, making IoT applications a challenge. FreeRTOS helps solve this problem by providing the kernel to run low power devices as well as software libraries that make it easy to securely connect to the cloud or other edge devices, so you can collect data from them for IoT applications and take action.

FreeRTOS Features

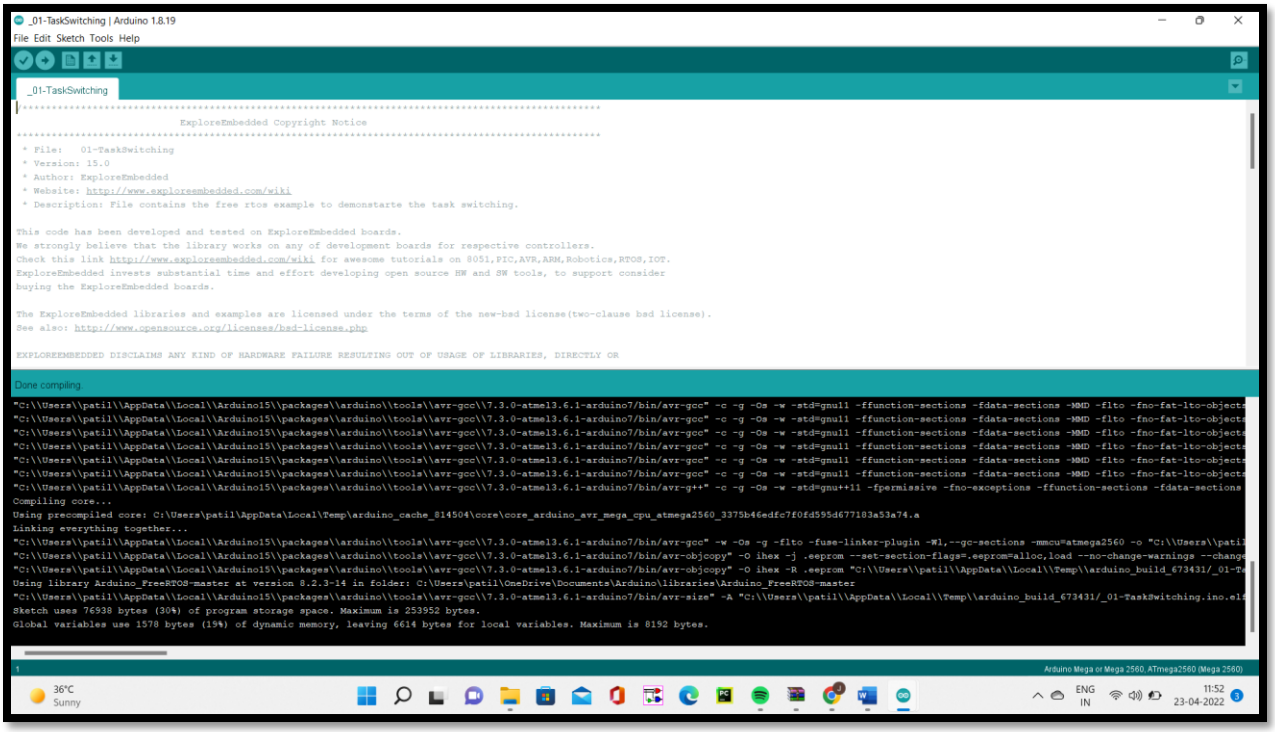
1. Highly configurable kernel.
2. Easy to use APIs for creating, configuring, and scheduling tasks.
3. Mutex and Semaphore for synchronization.
4. Support for thread-safe message queues.
5. Software timers.
6. Event groups to notify the occurrence of an event.
7. Memory management to maintain heap memory.

Procedure:

1. Download the Free RTOS library from internet and then include the library in Arduino IDE
2. Go to Files -> Examples -> FreeRTOS
3. We can see all the built in files of free RTOS.



Then run the first example to ensure its working:



Post-Lab Questions:

1. Compare the features of Free RTOS with RT-Linux w.r.t scheduling, Inter-process communication, Memory management and Interrupt Latency.
2. Explain briefly the different RTOSes supported by Arduino.

9073 Arpita Kar

PAGE No. 1
DATE / /

EXP - 7

Q1.]

> While size is a major difference between FreeRTOS and RTLinux, so are the functions they provide FreeRTOS is a very minimalistic operating system. It provides only some basic scheduling inter process communication (IPC) & semaphores for synchronization.

> RT Linux on the other hand can provide all the things that a normal linux distribution can. This includes networking support, graphical environment, web server and much more. These features, however run in the Linux Kernel. That means that they are not strictly speaking part of the real-time system.

Q.2.]

> Free RTOS is a very popular RTOS which supports either preemtive or co-operative scheduling, depending on how its configured. It supports wide range of microcontroller in the Arduino family such as ATmega 328, ATmega 128e, etc.

> Helios is a co-operative scheduled OS that's been designed around ease of use. by opting for co-operative scheduling. Helios should be easier then some other OS's to write safe code for.

9073 Arpita Kar

PAGE No. 2
DATE / /

EXP-7

> Simba is an embedded programming platform platform which sets it apart from the other choices in the piece because it &p supports more that just task management. Type of scheduling used is pre-emptive or cooperative.

> Co-op threads is a cooperative schedule that was designed to be light weight so that till run on 8-bit micro-controller.

8. Multi-tasking using FreeRTOS

Course, Subject & Experiment Details

Academic year	2021 – 2022	Estimated Time	02 Hours
Course	T.E. (ECS)	Subject Name	Embedded systems and RTOS
Semester	VI	Chapter Title	FreeRTOS
Experiment Type	Coding	Subject Code	ECC 601

Aim & Objective of Experiment

To demonstrate multi-tasking (using 2 tasks) using FreeRTOS

Theory:

Each executing program is a **task** (or thread) under control of the operating system. If an operating system can execute multiple tasks in this manner it is said to be **multitasking**.

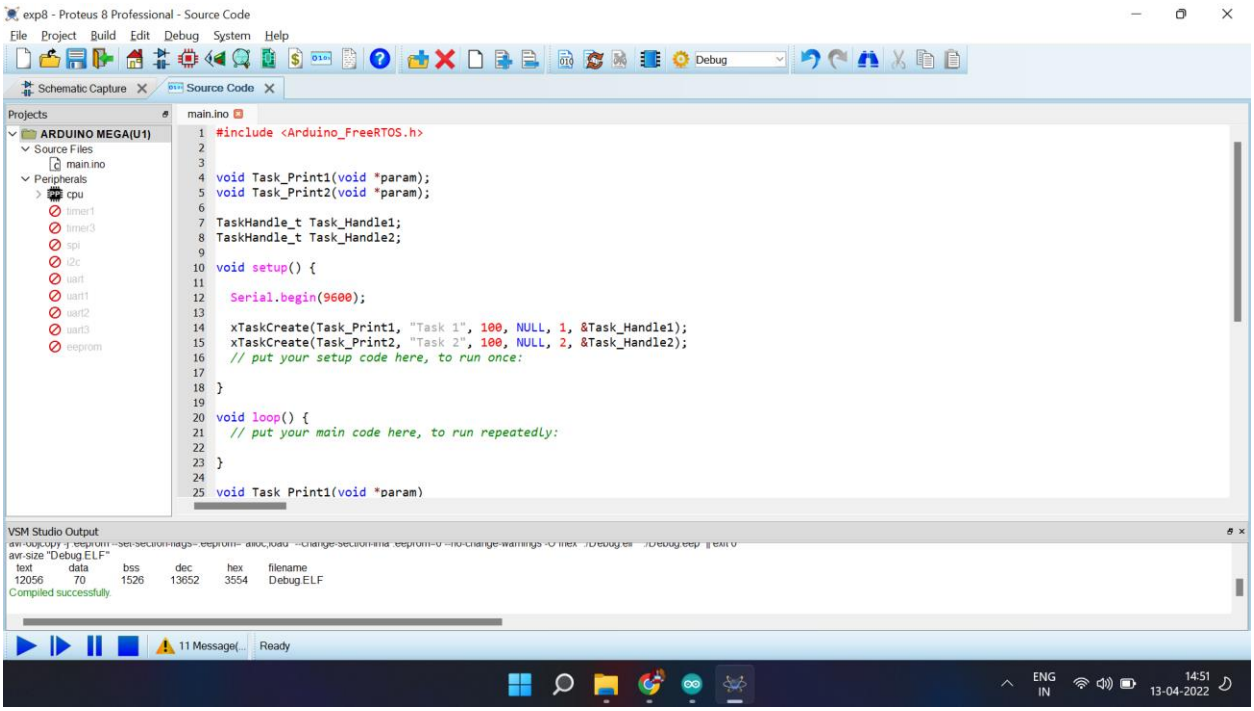
The use of a multitasking operating system can simplify the design of what would otherwise be a complex software application:

- The multitasking and inter-task communications features of the operating system allow the complex application to be partitioned into a set of smaller and more manageable tasks.
- The partitioning can result in easier software testing, work breakdown within teams, and code reuse.
- Complex timing and sequencing details can be removed from the application code and become the responsibility of the operating system.

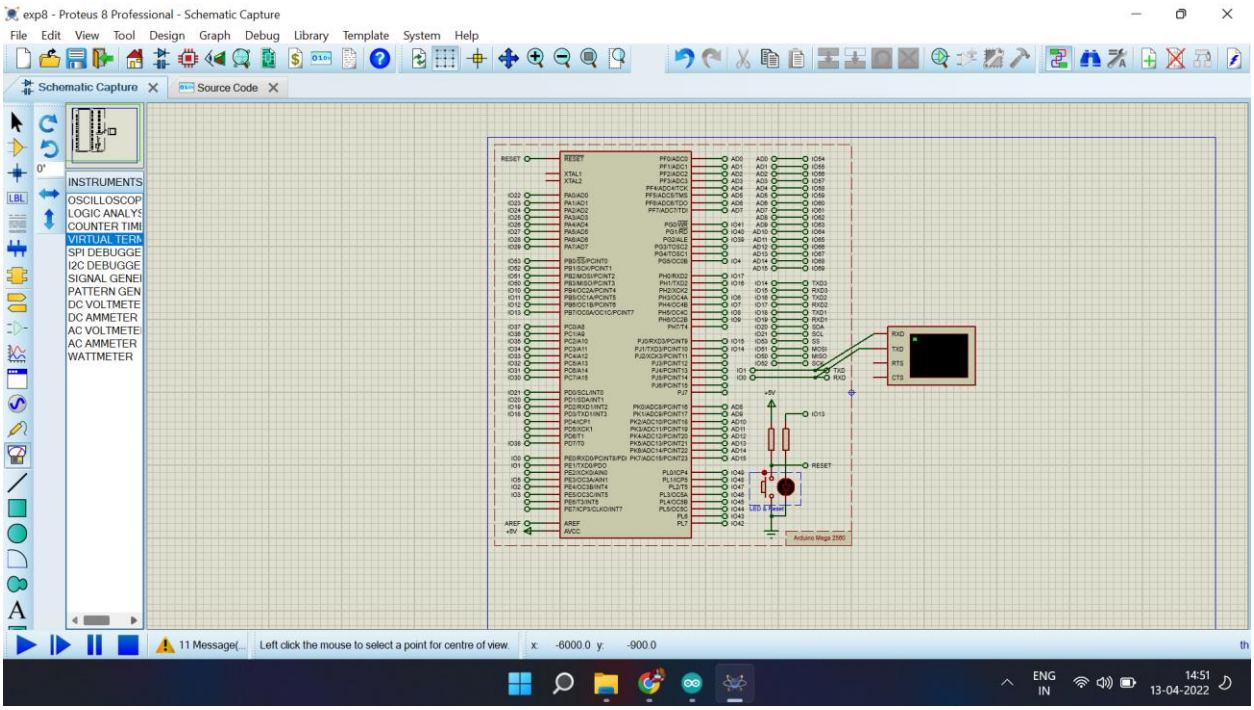
Algorithm:

```
#include <Arduino_FreeRTOS.h>
void Task_Print1(void *param);
void Task_Print2(void *param);
TaskHandle_t Task_Handle1;
TaskHandle_t Task_Handle2;
void setup() {
  Serial.begin(9600);
  xTaskCreate(Task_Print1, "Task 1", 100, NULL, 1, &Task_Handle1);
  xTaskCreate(Task_Print2, "Task 2", 100, NULL, 2, &Task_Handle2);
}
void loop() {
}
void Task_Print1(void *param) {
  (void) param;
  while(1){
    Serial.println("Task 1");
    vTaskDelay(1000/portTICK_PERIOD_MS);
  } }
void Task_Print2(void *param) {
  (void) param;
  while(1) {
    Serial.println("Task 2");
    vTaskDelay(1000/portTICK_PERIOD_MS);
  }
}
```

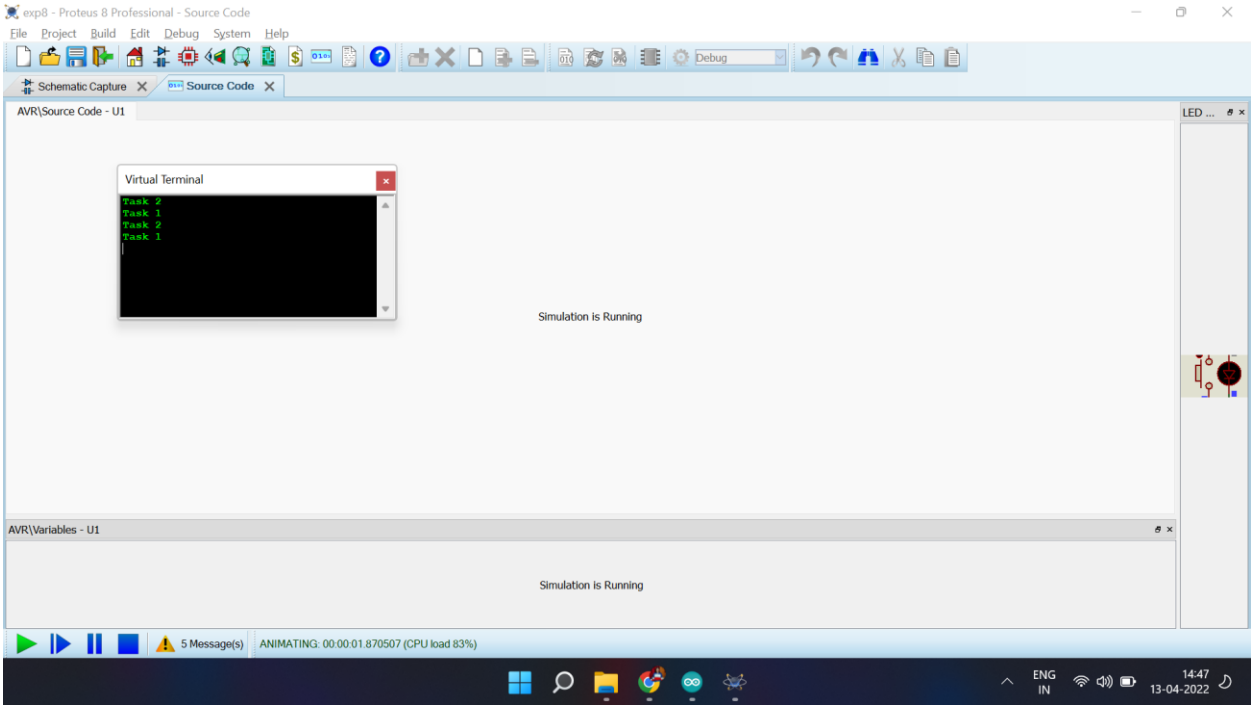
Code:



Proteus:



Output:



Post-Lab questions

1. Explain the task related functions of FreeRTOS
2. Explain the VTaskDelay() function

9073 Arpita Kar

PAGE No. 1
DATE / /

EXP-8

Q.1.] There are various Task relation functions.
> Task creation function's consists of Task Create - used to create a new task, Task State x, x Task Create Restricted State x - create a new memory protection unit restricted task and add it into the list, v Task Delete - to remove a task from RTOS kernals managements.
> Task control functions consists of Task controlling functions such as to provide delay, v Task Delay, v Task Delay Until(), To set priority of task using - U x Task Priority Get(), v Task Priority Set(), v Task Suspend(), v Task Resume()
> Task utility related functions consists of
a) x Task Get Current handler: Returns the handler to the curntly running task.
b) x Task Get Idle Task Handle.
c) e Task Get State
d) PC Task Get Name
e) x Task Get Handle
f) x Task Get Tick Count
g) v Task List
h) v Task Get Run Time Counter.

Q.2.] v Task Delay function specifies a time at which the task wishes to unblock relative to the time at which v Task Delay() is called. For eg, Specifying a block period for 100 ticks will cause

9073 Arpita Kar

PAGE No. (2)
DATE / /

EXP-8

the task to unblock 100 ticks after the function is called.

> parameters would include x Ticks to Delay() - the amount to time, in tick periods that the calling task should block.

9. Task-related functions using FreeRTOs

Course, Subject & Experiment Details

Academic year	2021 – 2022	Estimated Time	02 Hours
Course	T.E. (ECS)	Subject Name	Embedded systems and RTOS
Semester	VI	Chapter Title	FreeRTOS
Experiment Type	Coding	Subject Code	ECC 601

Aim & Objective of Experiment

To demonstrate the usage of vTaskSuspend, vTaskResume and vTaskDelete functions of FreeRTOS

Theory:

vTaskSuspend(): This function is used to Suspend a task, the suspended remains in the same state util it is resumed. For this, we need to pass the handle of the tasks that needs to be suspended. Passing NULL will suspend own task.

vTaskResume(): This function is used to resume a suspended task. If the Resumed task has higher priority than the running task then it will preempt the running task or else stays in ready state

vTaskDelete(): This function is used to delete as task. We need to pass the taskHandle of the task to be deleted.

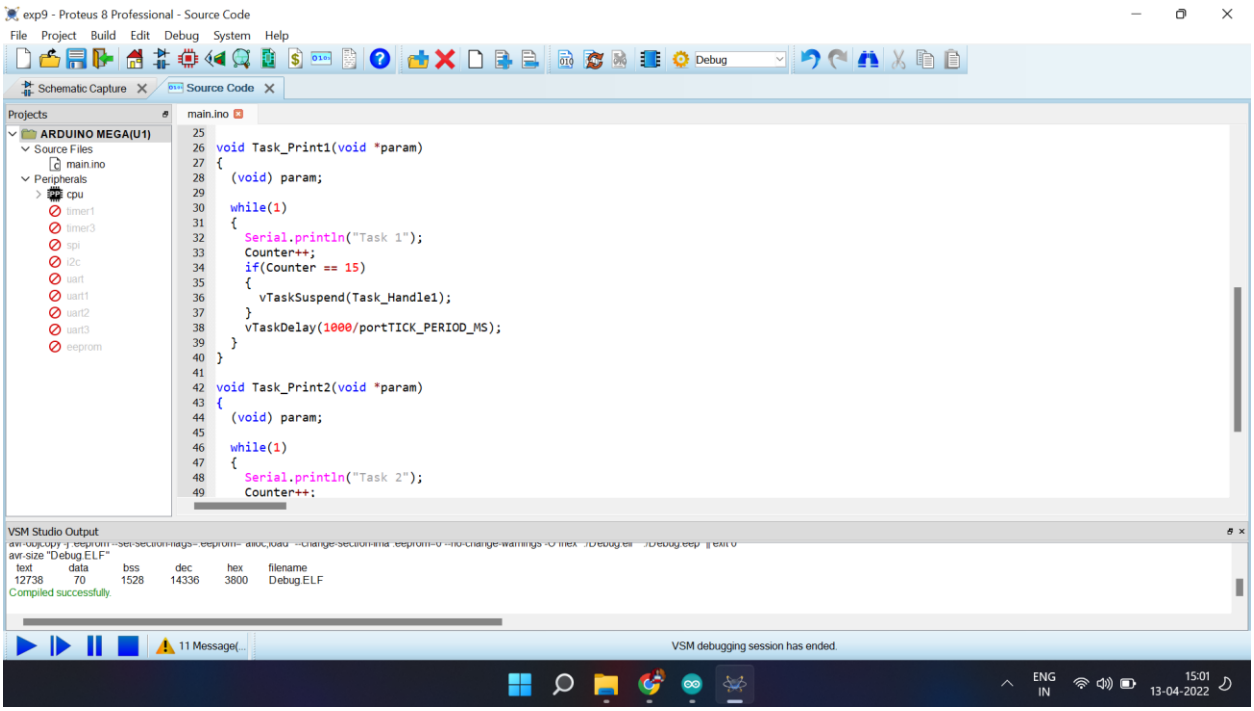
To delete the own task we should pass NULL as the parameter.

Algorithm:

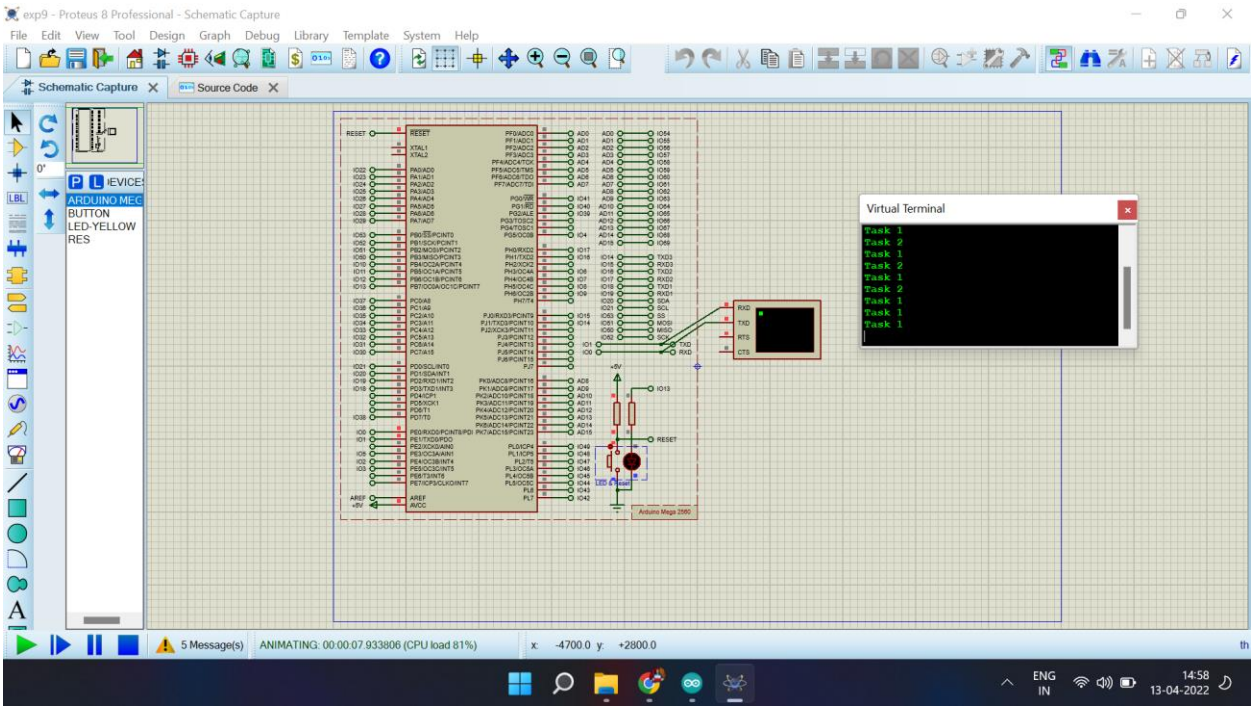
```
#include <Arduino_FreeRTOS.h>
void Task_Print1(void *param);
void Task_Print2(void *param);
TaskHandle_t Task_Handle1;
TaskHandle_t Task_Handle2;
int Counter = 0;
void setup() {
  Serial.begin(9600);
  xTaskCreate(Task_Print1, "Task 1", 100, NULL, 1, &Task_Handle1);
  xTaskCreate(Task_Print2, "Task 2", 100, NULL, 1, &Task_Handle2);
  // put your setup code here, to run once:
}
void loop() {
  // put your main code here, to run repeatedly:
}
void Task_Print1(void *param)
{
  (void) param;
  while(1)
  {
    Serial.println("Task 1");
    Counter++;
    if(Counter == 15)
    {
      vTaskSuspend(Task_Handle1);
```

```
    }
    vTaskDelay(1000/portTICK_PERIOD_MS);
  }
}
void Task_Print2(void *param)
{
  (void) param;
  while(1)
  {
    Serial.println("Task 2");
    Counter++;
    if(Counter == 10)
    {
      vTaskDelete(Task_Handle2);
    }
    vTaskDelay(1000/portTICK_PERIOD_MS);
  }
}
```

Code:



Proteus and Output:



Postlab questions

1. Explain the instances when tasks may need to be temporarily suspended and resumed later.
2. What is the possible disadvantage of deleting tasks?

9073 Arpita Kar

EXP-9

Q1.]

> If process A is using a shared file & process B is requesting for the same shared file, then process B would be sent into temporary suspension.
> If a process waited for an event to happen for a longer period of time, the process would be suspended.
> Tasks would be suspended if a higher priority task needs to be executed.
> If a task is waiting for a external interrupt or an event to happen.

Q. 2.]

1) During deletion process, kernel terminates the task and frees the memory by deleting the task's TCB and stack.
2) However, when task executes, they can acquire memory or access resources with other kernel objects. If the task is detected in correctly, the task might not release the resources, which can lead to:
   (a) A corrupt data structure, due to incomplete write operations.
   (b) An unreleased semaphore, which will not be available for other tasks to acquire it.
   (c) An inaccessible data structure, due to unreleased semaphore.
   (d) premature detection of tasks can lead to memory & or resource leaks

10. Inter-process communication using FreeRTOS

Course, Subject & Experiment Details

Academic year	2021 – 2022	Estimated Time	02 Hours
Course	T.E. (ECS)	Subject Name	Embedded systems and RTOS
Semester	VI	Chapter Title	FreeRTOS
Experiment Type	Coding	Subject Code	ECC 601

Aim & Objective of Experiment

To demonstrate the usage of a Mutex using FreeRTOS

Theory:

Inter Process Communication (IPC) refers to a mechanism, where the operating systems allow various processes to communicate with each other. This involves synchronizing their actions and managing shared data.

Mutexes are binary semaphores that include a priority inheritance mechanism. Whereas binary semaphores are the better choice for implementing synchronisation (between tasks or between tasks and an interrupt), mutexes are the better choice for implementing simple mutual exclusion (hence 'MUT'ual 'EX'clusion).

When used for mutual exclusion the mutex acts like a token that is used to guard a resource. When a task wishes to access the resource it must first obtain ('take') the token. When it has finished with the resource it must 'give' the token back - allowing other tasks the opportunity to access the same resource.

Note that Binary semaphores and mutexes are very similar but have some subtle differences: Mutexes include a priority inheritance mechanism, binary semaphores do not. This makes binary semaphores the better choice for implementing synchronisation (between tasks or between tasks and an interrupt), and mutexes the better choice for implementing simple mutual exclusion.

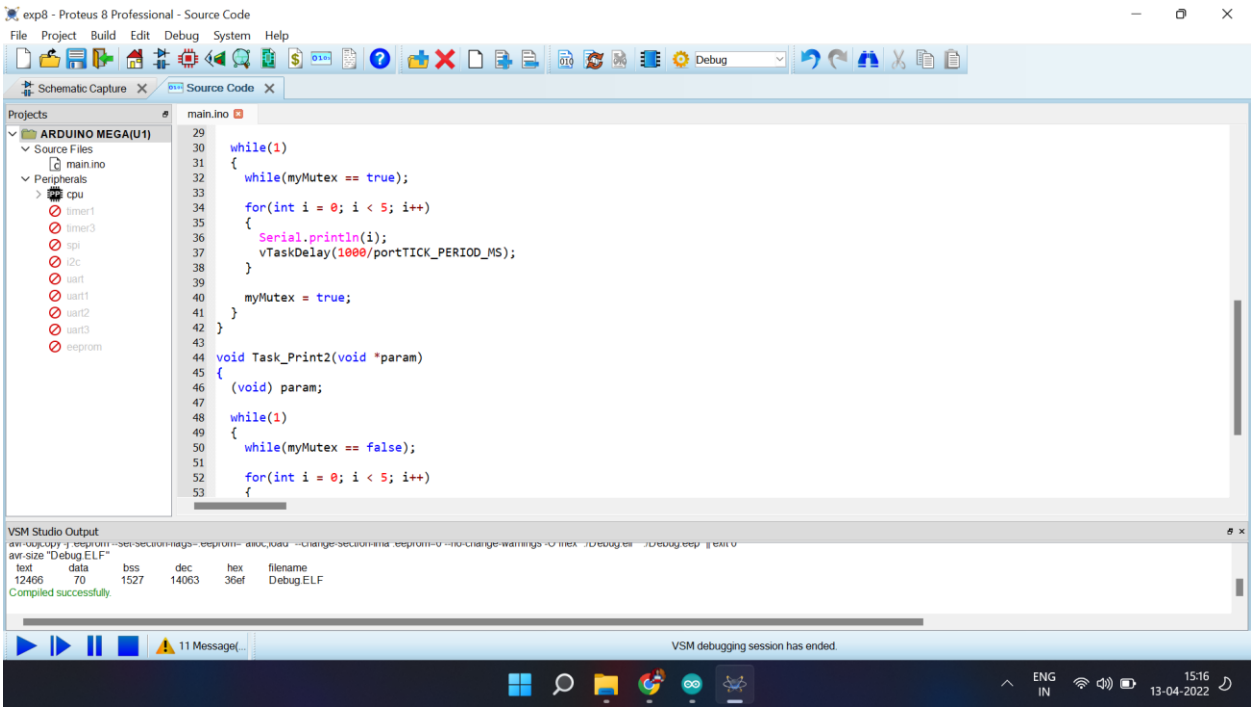
A semaphore can be used in order to control the access to a particular resource that consists of a finite number of instances.

Algorithm:

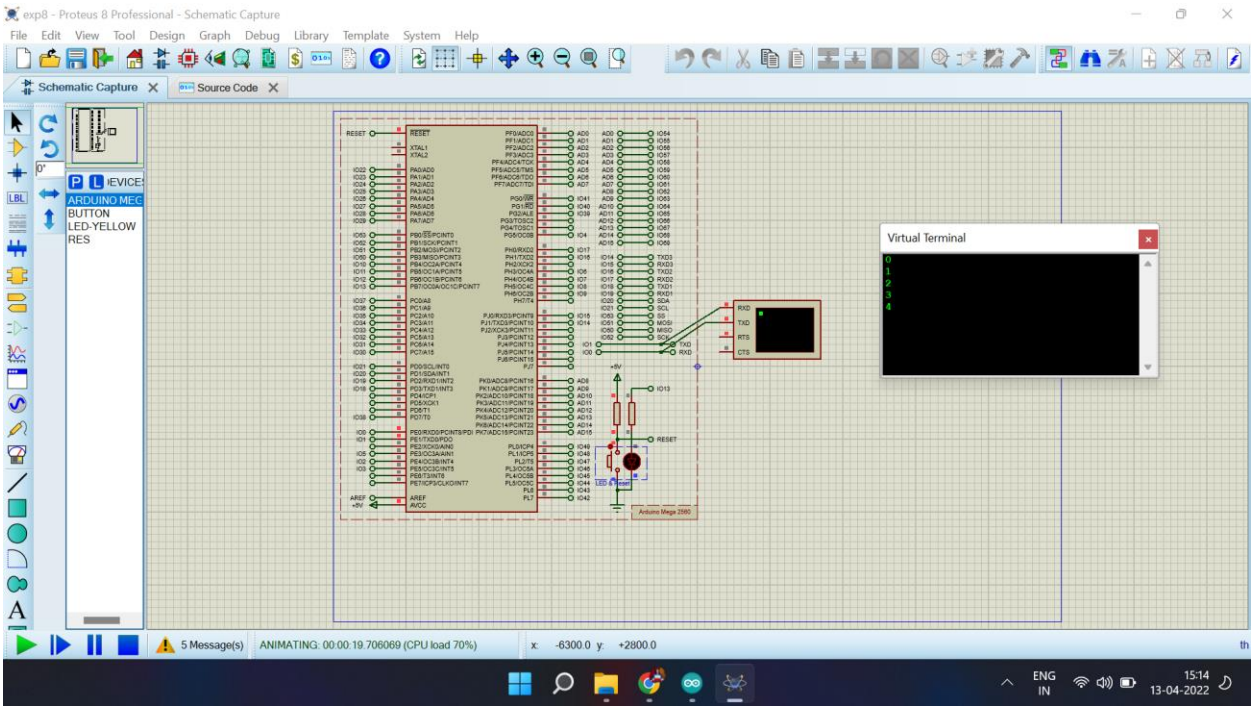
```
#include <Arduino_FreeRTOS.h>
void Task_Print1(void *param);
void Task_Print2(void *param);
TaskHandle_t Task_Handle1;
TaskHandle_t Task_Handle2;
volatile boolean myMutex = false;
void setup() {
  Serial.begin(9600);
  xTaskCreate(Task_Print1, "Task 1", 100, NULL, 1, &Task_Handle1);
  xTaskCreate(Task_Print2, "Task 2", 100, NULL, 1, &Task_Handle2);
  // put your setup code here, to run once:
}
void loop() {
  // put your main code here, to run repeatedly:
}
void Task_Print1(void *param)
{
  (void) param;
  while(1)
  {
    while(myMutex == true);
    for(int i = 0; i < 5; i++)
    {
      Serial.println(i);
      vTaskDelay(1000/portTICK_PERIOD_MS);
```

```
    }
    myMutex = true;
  }
}
void Task_Print2(void *param)
{
  (void) param;
  while(1)
  {
    while(myMutex == false);

    for(int i = 0; i < 5; i++)
    {
      Serial.println(i);
      vTaskDelay(1000/portTICK_PERIOD_MS);
    }
    myMutex = false;
  }
}
```

Code:



Proteus and Output:



Postlab questions:

1. Explain the various IPC techniques supported by FreeRTOS
2. Clearly describe the usage of a queue IPC

9073 Arpita Kar

PAGE No. 1
DATE / /

EXP - 10

Q1.]

(a) Different IPC techniques supported by RTOS are:

i) Semaphore
ii) Mutex
iii) Message queue
iv) Mailbox
v) Event registes
vi) Signals.

(b) Semaphore is a kernal object provided by RTOS & can be used in several ways. It is a form of inter process com^n.

(c) Mutex is a mutual exclusion lock. Only one Mutex can holod the lock Mutex are used to protect the data or other resources from concurrent access.

(d) A message queue is a linked list of message stored within the kernel and identified by a message queue identifier. All processes can exchange information through common system message queue.

Q.2.]

> A message queue is a linked list of message stored within the kernel and identified by a message queue identifier. A new queue is created or an existing queue is opend by msgget().

> New messages are added to the end of the queue by msgsnd(). Every message has a +ve long integer type field, a non-ve

Exp-10

length of the actual data bytes all of which are specific to msgrcv().

> Each message is given an identification or type, so that each process can select the appropriate messages. Process must share a common key in order to gain access to the queue itself.

Report
On
Case Study

Microwave

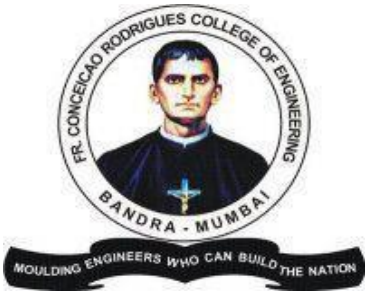
By

Sadiya Shaikh (9021)
Arpita Kar (9073)

Semester 6/ TE ECS

Professor

Prof. Sapna Prabhu



Department of Electronics and Computer Science

Fr Conceicao Rodrigues College of Engineering

Bandra, Mumbai - 400 050

University of Mumbai

(AY 2021-22)

I. INTRODUCTION

A microwave oven is an electric oven that heats and cooks' food by exposing it to electromagnetic radiation in the microwave frequency range.

Microwave ovens have a limited role in professional cooking, because the boiling-range temperatures of a microwave oven will not produce the flavorful chemical reactions that frying, browning, or baking at a higher temperature will.

However, such high heat sources can be added to microwave ovens in the form of a convection microwave oven.

It is a soft real time system.

II. FEATURES

1. We can grill, roast, bake and defrost food items.
2. Safe for Children
3. Variable Heat Settings
4. Low Energy Consumption
5. No Change in Taste and Nutrition

III. PARTS OF MICROWAVE

1. Transformer
2. Magnetron
3. Waveguide
4. Cooling fan
5. The rotating disc
6. 7-segment display
7. Input keypads

IV. WORKING

Inside the strong metal box, there is a microwave generator called a magnetron. A magnetron takes electrical energy and converts it into RF (Radio Frequency) energy.

When you start cooking, the magnetron takes electricity from the power outlet and converts it into high-powered radio waves.

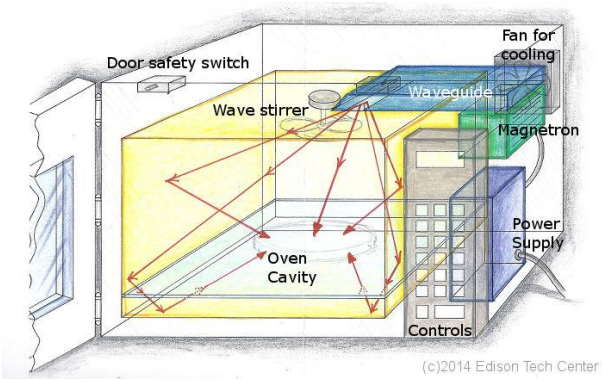
The magnetron blasts these waves into the food compartment through a channel called a wave guide.

The food sits on a turntable, spinning slowly round so the microwaves cook it evenly.

The microwaves bounce back and forth off the reflective metal walls of the food compartment. When the microwaves reach the food itself, it penetrates inside the food. As they travel through it, they make the molecules inside it vibrate more quickly.

Vibrating molecules have heat so, the faster the molecules vibrate, the hotter the food becomes.

Thus the microwaves pass their energy onto the molecules in the food, rapidly heating it up.



V. EMBEDDED SYSTEM IN MICROWAVE

Microwave ovens contain embedded systems and not operating systems. An OS handles a large framework of commands and functions, which is required by a computer with multiple programs.

An embedded system, on the other hand, performs a very specific function. These systems contain a single program that executes all the functions required.

The embedded system in a microwave oven works as a command device. It is designed to take directions from the keypad and turn them into commands.

VI. HARDWARE DESIGN

The AC power line voltage is converted down and regulated to 12 V and 5 V in the power supply module. The 5 V supply is the main supply for the whole system including MCU, I/O, display and rotary encoder. However, the buzzer circuit is driven by 12 V.

When an event triggered by the end-user through input keys or rotary encoder is decoded in MCU and the corresponding message or system status is shown in the 7-segment display as confirmation or indication.

The status of the oven door, open or close is monitored periodically as one of the error signals (high priority interrupt) for system protection.

A mechanical switch is used as a reliable method to cut off the ground connection for grill and microwave power stages when the oven door is open.

Different sound patterns generated from the buzzer are used as alert signals when fault conditions are detected.

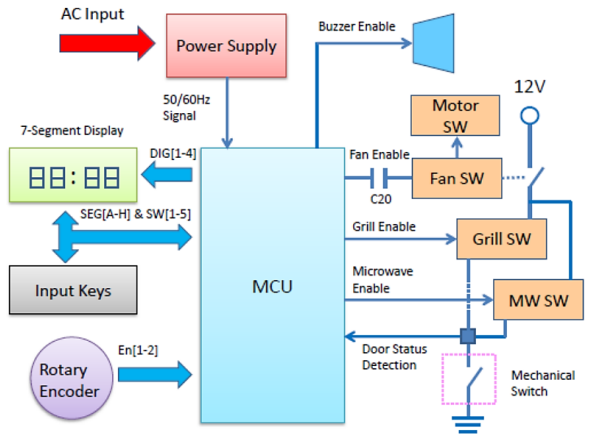


Figure 1. System block diagram

VII. SOFTWARE DESIGN

The main program continually executes a series of subroutines after system initialization to control the system according to user's inputs and keep track of all conditions for safety operation.

One of the Timer modules is configured in Output Compare mode to generate a periodic timing which is used as the core time base component for various software timers.

An individual counter variable is assigned for each software timer.

It counts up in the Timer interrupt routine and resets to zero when it reaches a predefined threshold.

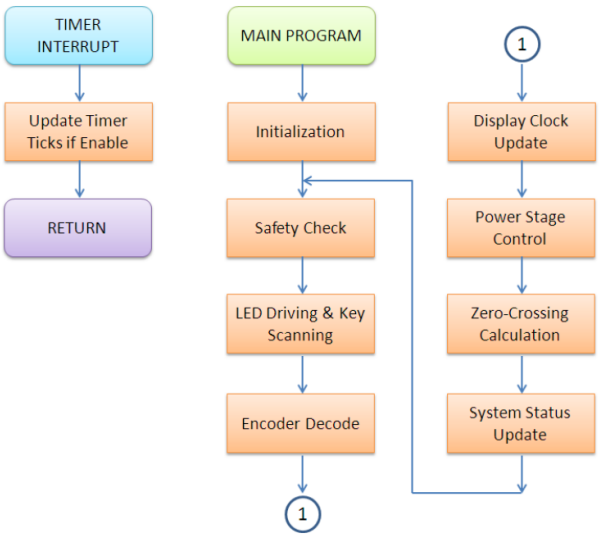


Figure 2. Main program flow

VIII. CONTROLLING MAGNETRON

The simplest is a relay that turns the magnetron on and off as required.

The newer technique includes power saving mode were an inverter proportionally turns the power down if less power is required.

IX. HIGH PRIORITY INTERRUPT

My friend insists on not opening a microwave door before it has stopped beeping at the end of its timer. He says it's to allow the magnetron to power down & the microwaves dissipate. Is there any truth to this?

- NO. As soon as the door opens, the interlock opens and a high priority interrupt is generated.
- The controller responds to this high priority interrupt and shuts down the magnetron. The high-voltage transformer for the magnetron is turned off immediately.
- The magnetron's high-voltage power supply discharges in less than 0.005 seconds. This is before the door can open more than a fraction of an inch.
- There is no residual power left to keep the magnetron operating.

X. REFERENCE

[1] https://www.quora.com/My-friend-insists-on-not-opening-a-microwave-door-before-it-has-stopped-beeping-at-the-end-of-its-timer-He-says-its-to-allow-the-magnetron-to-power-down-the-microwaves-dissipate-Is-there-any-truth-to-this

[2] https://www.sarthaks.com/1383095/explain-microwave-details-with-block-diagram-also-explain-principle-heating-microwave

[3] https://patents.google.com/patent/US4001537A/en

[4] https://www.explainthatstuff.com/microwaveovens.html

[5] https://fccid.io/APYDMR0140/Block-Diagram/Microwave-Oven-block-diagram-302611.pdf

[6] https://www.mrright.in/ideas/appliances/microwave/microwave-oven-components-and-their-functions/

[7] https://www.slideshare.net/SangeethSb/using-8051-microcontroller-based-washing-machine-control-ppt?next_slideshow=32822700

[8] https://www.hunker.com/13407974/the-parts-of-a-microwave-oven

[9] https://fccid.io/A3LNQ9300/Block-Diagram/Block-Diagram-3277658

[10] https://www.hunker.com/13407974/the-parts-of-a-microwave-oven

[11] Bhushan Band (2019). "Development of Microcontroller Based Smart Electric Oven Syste". https://www.researchgate.net/publication/332670693_Development_of_Microcontroller_Based_Smart_Electric_Oven_System