Faculty of Information Technology

The Winder

Final report

Group No: 37

204129M	**Meegoda S.Y.**
204189U	Sanjaya B.H.
204197R	Senevirathne Y.M.N. N.
204042N	Dinusha K.M.

Supervisor's Name: Mr B.H. Sudantha
Dean/Senior Lecturer
Faculty of Information Technology
University of Moratuwa

Co-Supervisor Name: Ms I. A. Wijethunga
Faculty of Information Technology
University of Moratuwa

Date of Submission: 08/06/2022

……. ……

Mr. B.H. Sudantha Ms. I. A. Wijethunga

Table of contents

1. Introduction

The transformer is an amazing invention, invented by Otto Blathy, Miksa Deri, and Karoly Zipernowsky in 1885. With the help of this device, can transfer electrical energy from one circuit to another circuit or multiple circuits. This electrical energy is transferred between two circuits without using a metallic connection between two circuits.

Transformers are mostly used to convert AC high voltages into low voltages (step down transformer) and AC low voltages into high voltages (step-up transformers). In the present world, various transformers are used to perform various tasks. Single supply transformers, dual supply transformers, grounding transformers, more polyphase transformers, and solid-state transformers are some of the examples of various types of transformers. To perform various tasks transformers come in large-scale sizes to small-scale sizes. An electrical powerhouse transformer is an example of a large-scale size transformer, and a power supply unit's transformer is an example of a small-scale transformer.

Every transformer consisted of several main components such as copper coils, insulation guards, and laminated steel cores (EI laminations). Copper coils are used to transfer the electricity through the transformer and the insulation guard is used to insulate the coils to prevent electrical shorts. The laminated steel cores are used to reduce -flux leakage.

Nowadays large-scale transformers are winded by using huge machines, but the small-scale transformers are winded by electricians by themselves or using semi-automated machines which are used to wind only the copper coil.

2. Literature Survey

In the present world market, there are some machines which are similar to our product. Below list contains the machine list.

1. BM Fabs company transformer winding machine [4].
2. Synthesis winding technologies Pvt Ltd transformer winding machine [5].

In BM fabs transformer winding machine, it consists of several levers and wheels. This machine doesn't operate by using electrical power. When an electrician needs to wind a transformer using this machine, he needs to wind it by himself by using the levers in the machine. One of the major issues is this machine is that it isn't an automated machine, and it is unable to include the insulation tape while the rotation process is ongoing. And, when using this machine, the electrician must carry out the mathematical calculation by himself before winding the transformer.

Synthesis winding technologies Pvt Ltd also has released a transformer winding machine to the world market. This machine can output a completed transformer with both coil and insulation tape. This machine can wind coil diameter between 400mm - 650mm and coil length between 550mm -850mm. This machine can wind only large-scale transformers. Another issue with this machine Is, that if the hardware is used in this machine, the cost of this machine is expensive. And, this machine isn't suitable for winding small-scale transformers.

3. Aim and Objectives

3.1. Aim

Designing, developing, and automating every action in the manual transformer winding process and improving the efficiency and effectiveness of the process.

3.2. Objectives

- To increase the efficiency and effectiveness of the transformer winding process.
- To save the time which is committed to the winding process.
- To minimize the human errors, that occur in the transformer winding process.

4. Analysis and Design

4.1. Block diagram

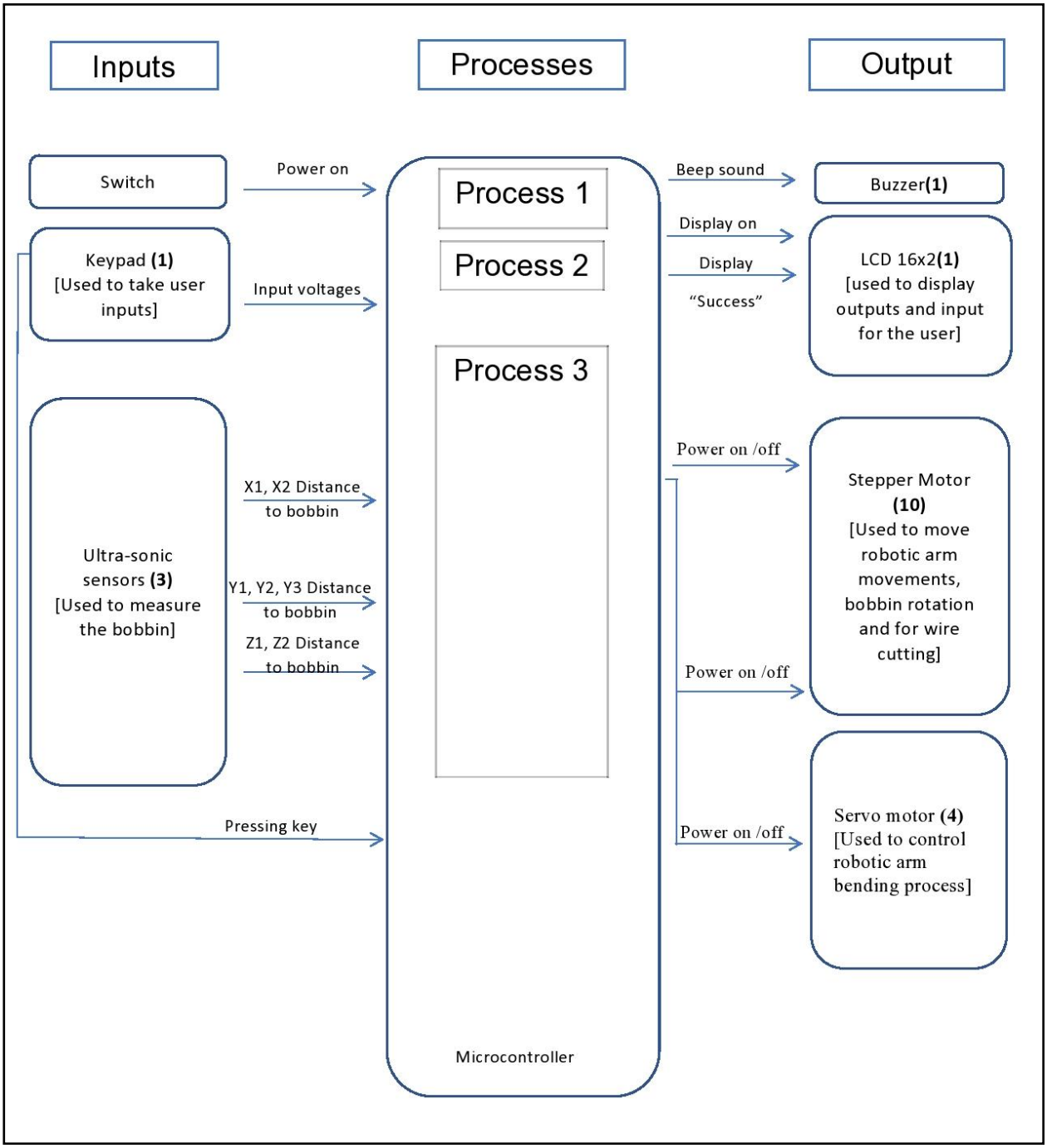

Figure 1-Block diagram

4.2. 3D Diagram

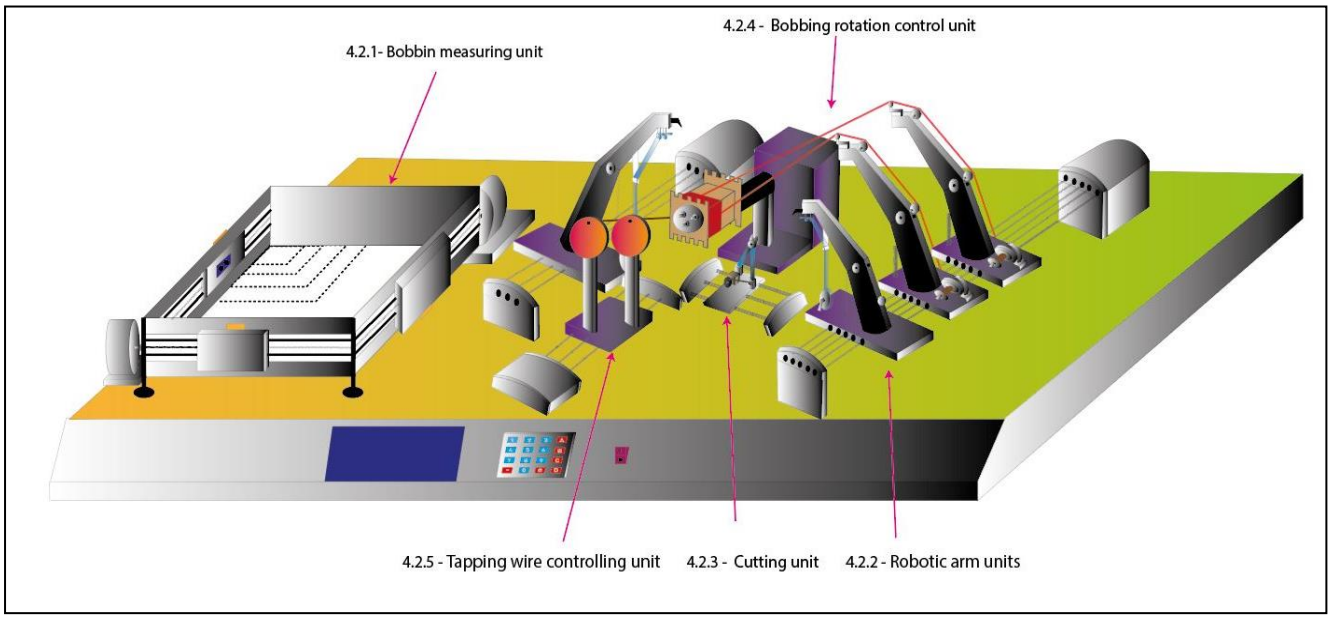

Figure 2-3D Model

This 3d graph shows 6 main subsections of the machine.

4.2.1 Bobbin measuring unit

4.2.2 Robotic arms unit

4.2.3 Cutting Unit

4.2.4 Bobbin rotation controlling unit

4.2.5 Tapping wire control unit

4.2.1. Bobbin measuring unit

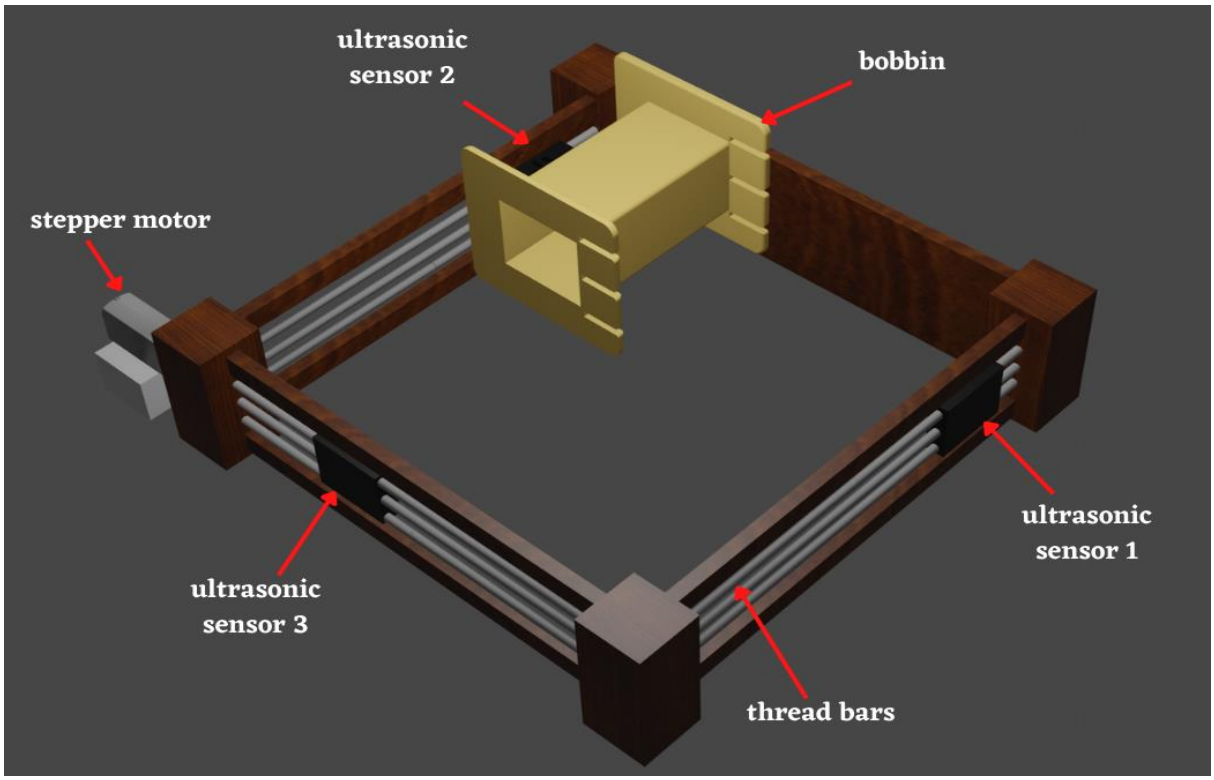

Figure 3-Bobbin measuring unit

When the user switches on the machine, he/she needs to input the primary voltage and secondary voltage via the keypad. After that, the user needs to input the number of tappings to the machine(if there is none user should input 0).

Ex:- If the user inputs 3 for several tappings, then he/she should input relevant 3 output voltages that are needed from the secondary coil.

After the input process is finished, the LCD screen will display the relevant core area size for a bobbin. Then the user must place the relevant bobbin on top of dotted lines on the machine which are marked on the body of the measuring unit(see *Figure 3-Bobbin measuring unit*). After placing the bobbin, the measuring unit takes 3 lengths(see *Figure 7-Y length* and see *Figure 8-Z length*) from the bobbin by using 3 ultrasonic sensors. In this unit ultrasonic sensor 3 is moving horizontally by using a stepper motor. These lengths are essential in calculating the core area(see

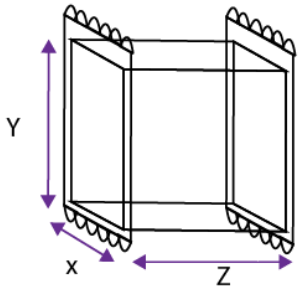

Figure 4-Bobbin measurements

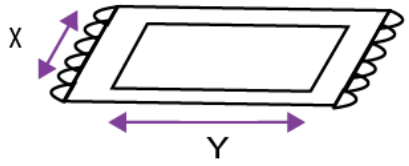

Figure 5-Core area

Figure 4-Bobbin measurements and *Figure 5-Core area*) and deciding robotic arm moving distances.

X lenth calculation(X = X1-X2-X3)

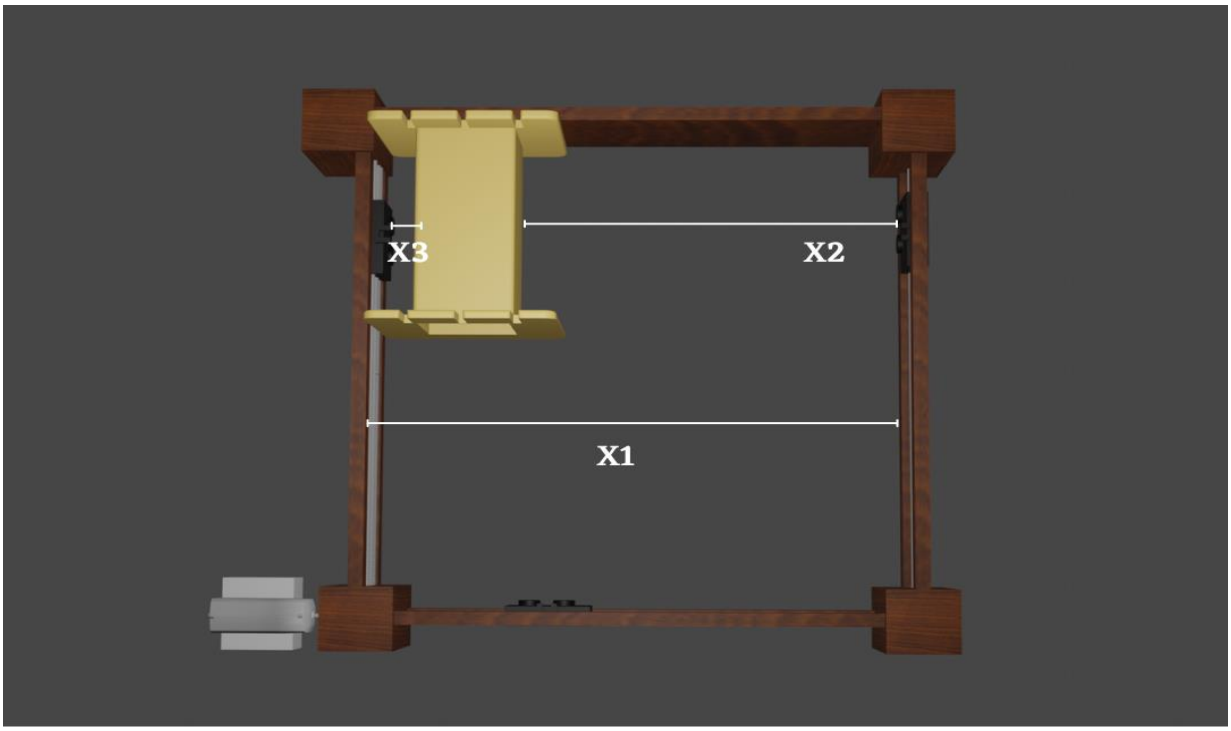

Figure 6-X length

Y lenth calculation(Y = Y1- Y 2- Y3)

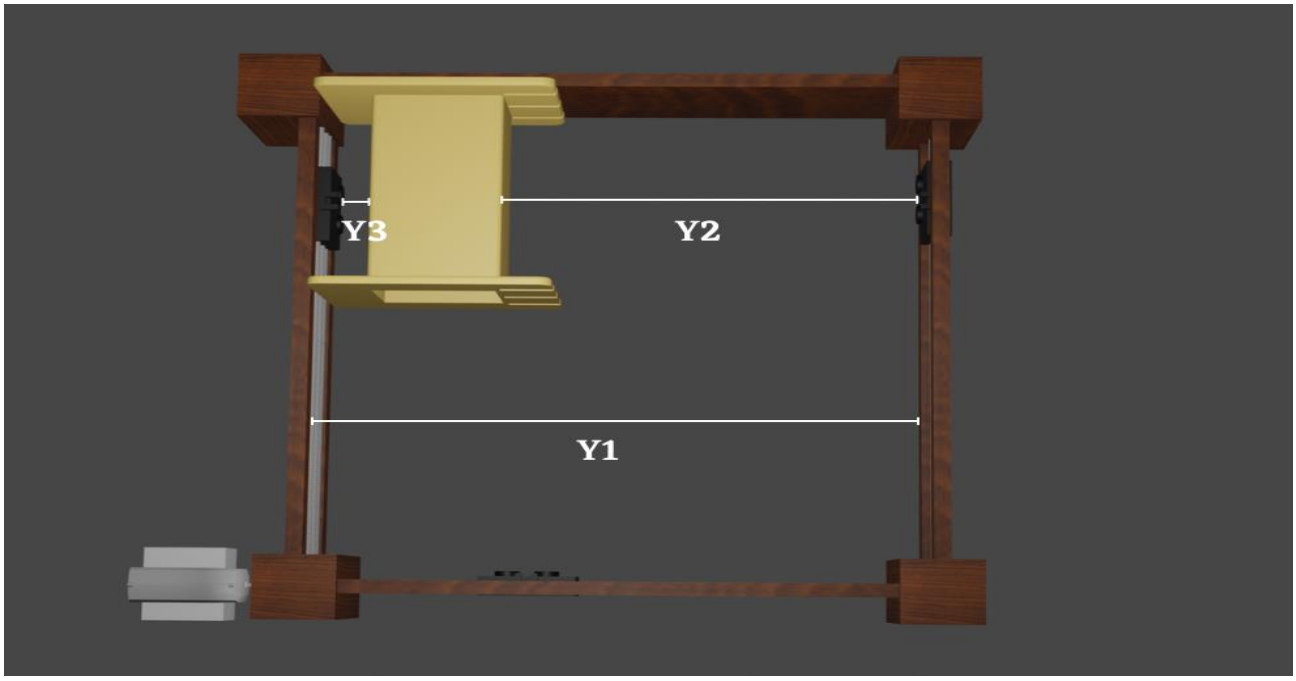

Figure 7-Y length

Z lenth calculation(Z = Z2- Z1)

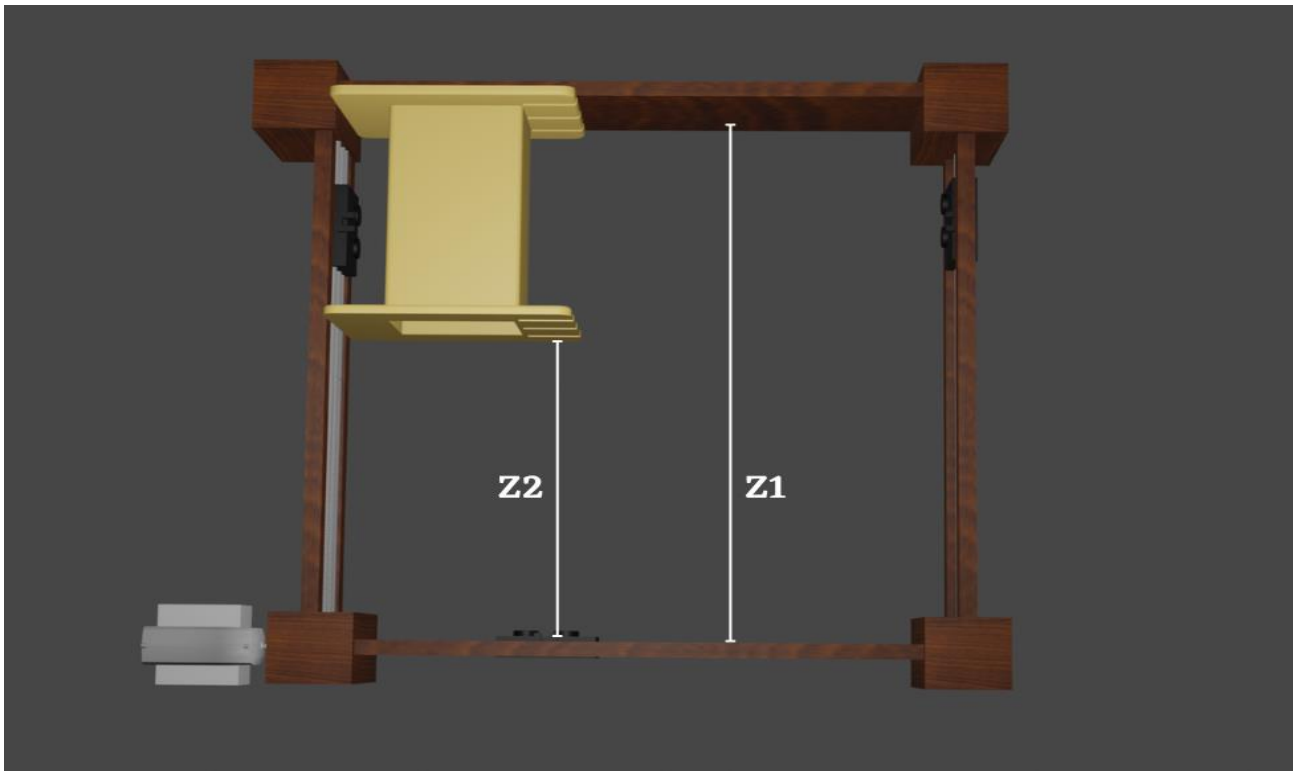

Figure 8-Z length

4.2.2. Robotic arms unit

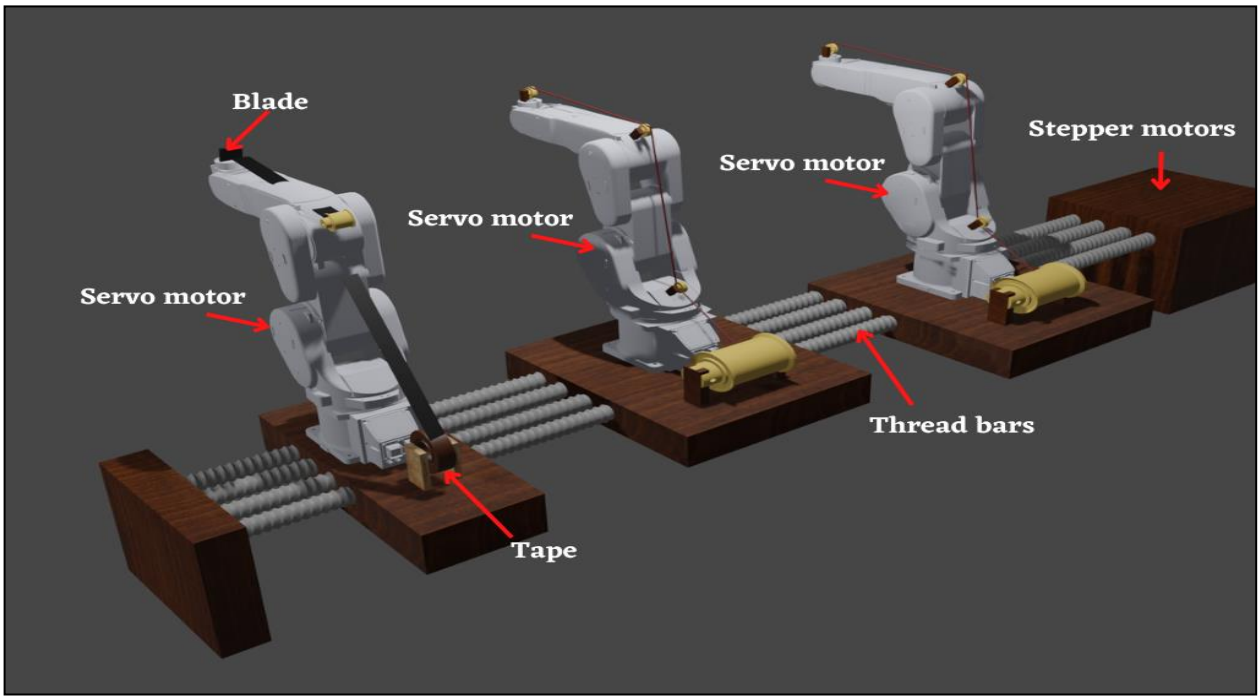

Figure 9-Robotic arm unit(Primary,Secondary arm and taping arm1)

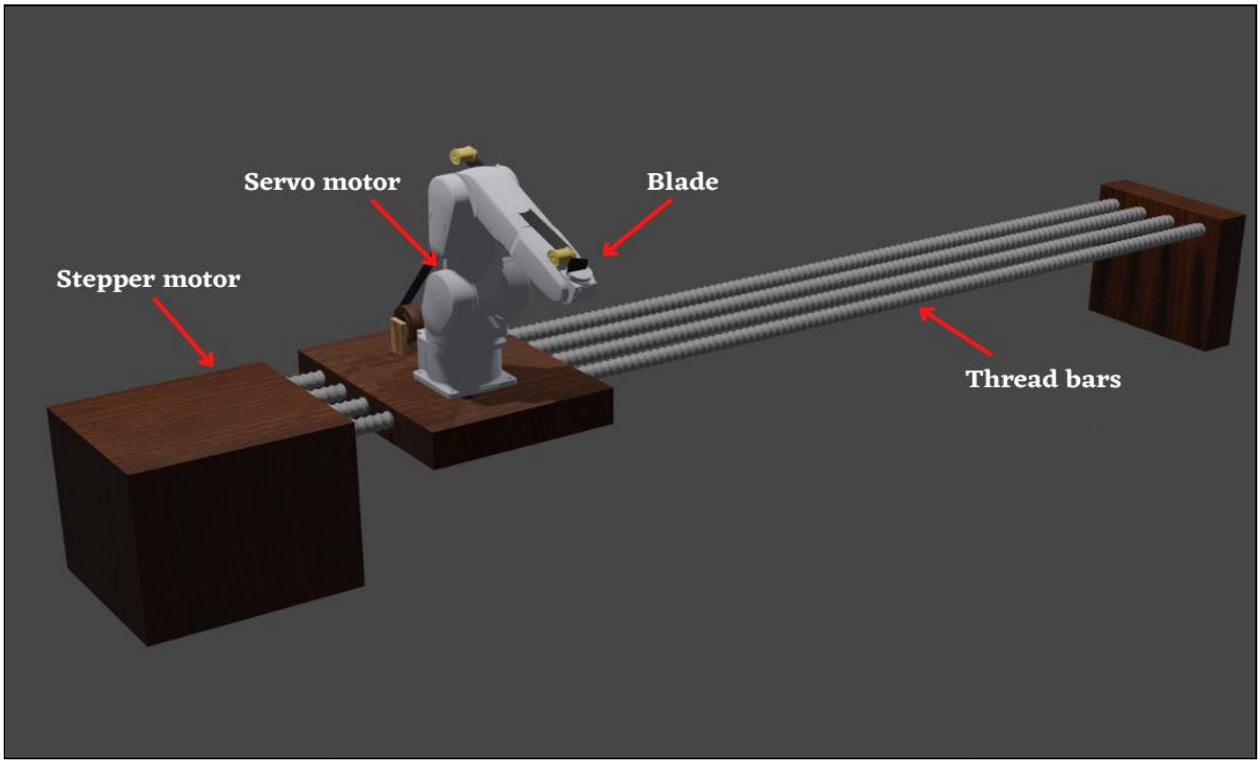

Figure 10- Robotic arm unit(taping arms)

This unit consists of two types of robotic arms.

1. Winding process controlling arms(see *Figure 9-Robotic arm unit(Primary,Secondary arm and taping arm1)*)
2. Taping controlling arms(see *Figure 10- Robotic arm unit(taping arms)*)

After the measuring part is completed, the LCD screen displays the relevant gauges for both primary and secondary coils. Then the user should insert the relevant primary and secondary coil yarn balls into coil holders. After inserting the coils, the user should insert the primary and secondary coils to the primary and secondary arms through the pulleys and need to fold the coils to the primary coil detainer and to the secondary coil detainer(detainers are named in 4. Bobbin rotation controlling unit **(**see *Figure 12- Bobbin rotation control unit***)**.

After inserting the coils accurately, the user needs to press the "#" key button to start the winding process. Then the primary arm starts to move from left side to right side on thread bars and 4. Bobbin rotation controlling unit starts the rotation.

While the rotation is going on, one of the taping arms is used to cover the primary coil with the insulation tape.

Once the primary coil completes the number of turns, it moves to the left side to cut the coil by using the cutter(see *Figure 11- Cutting unit*). Then it moves to the right side corner through thread bars to give space to the secondary arm. The secondary arm also completes the above winding process as well as the primary arm.

4.2.3. Cutting Unit

This separate unit is designed to cut the coils from the primary and secondary arms after completing the windings. With the help of bars in this unit, the cutter can be moved to any side of the bobbin.

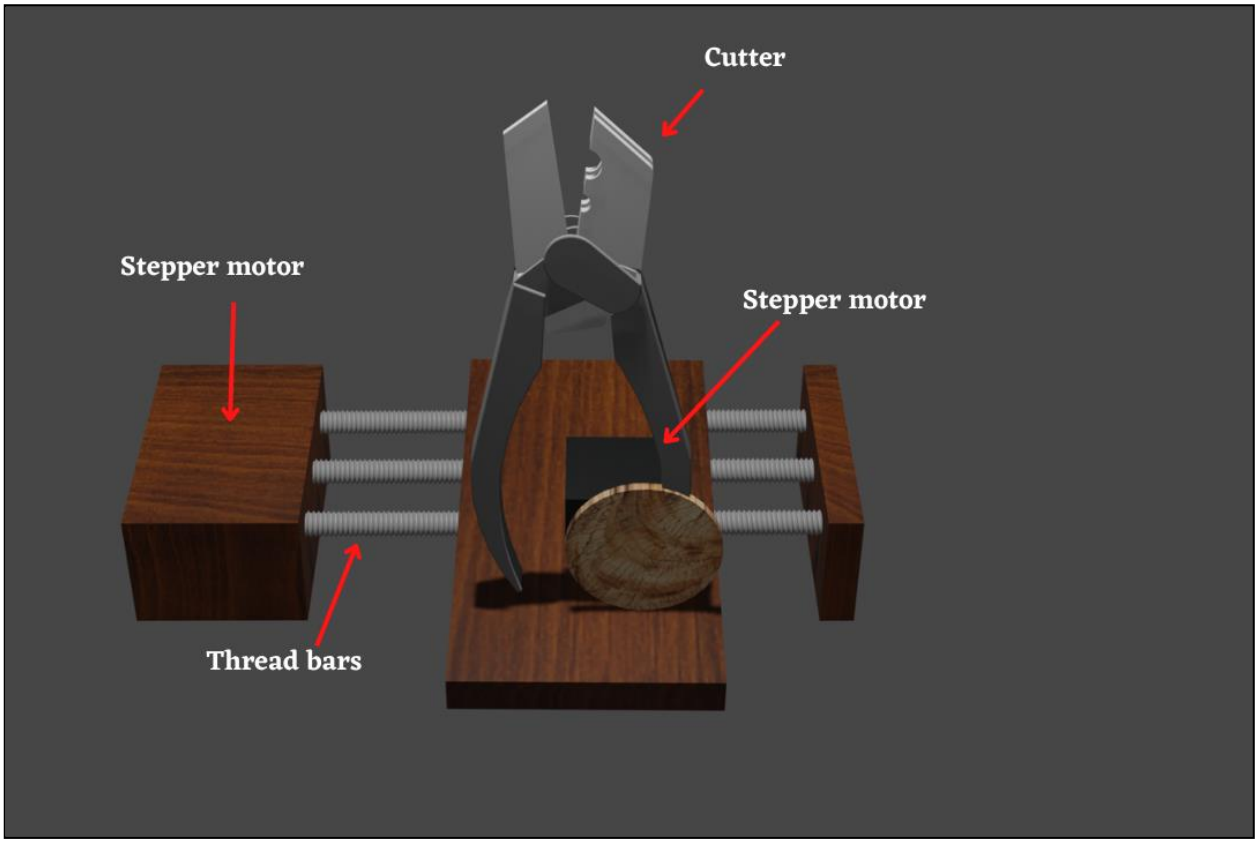

Figure 11- Cutting unit

4.2.4. Bobbin rotation controlling unit

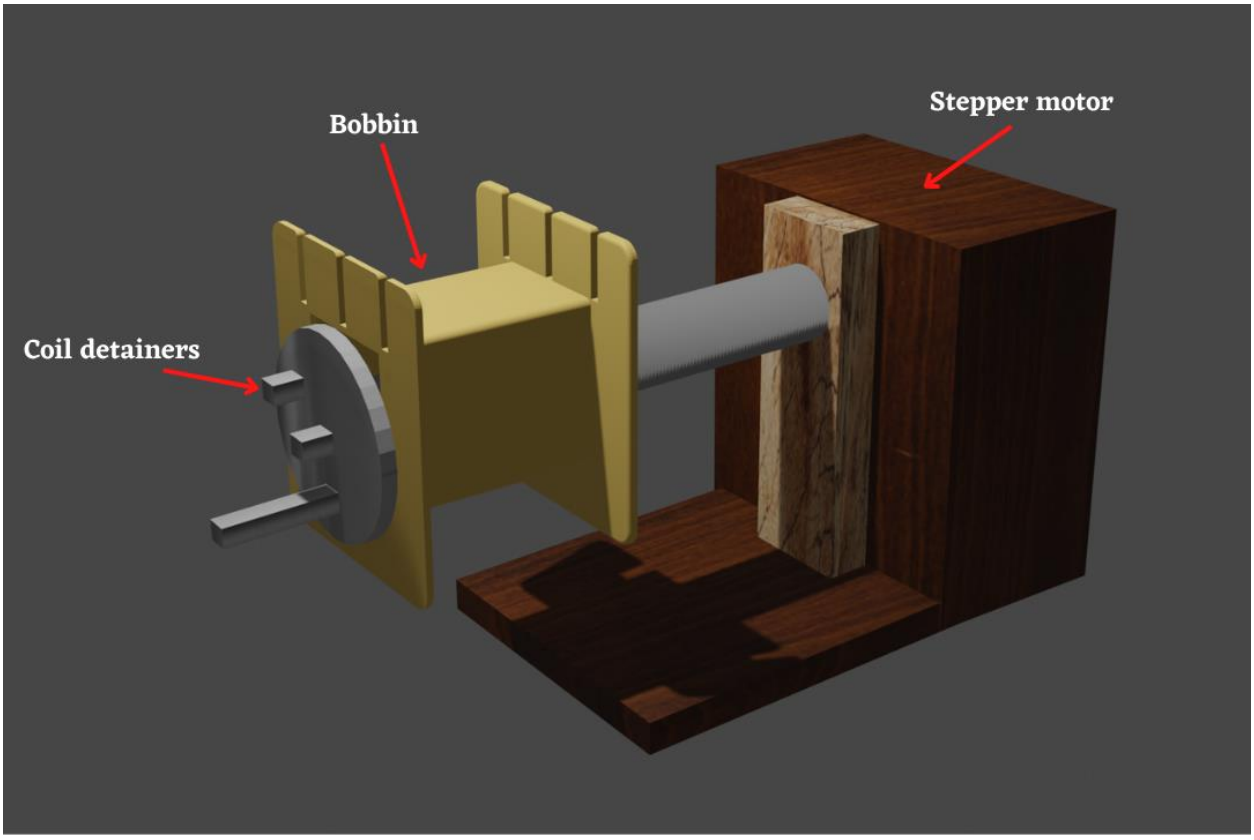

Figure 12- Bobbin rotation control unit

This separate unit controls the total winding process in the machine. This unit is consisted of primary, secondary, tapping detainers and the motor unit. With the help of this unit, rotation of the bobbin is processed by using a stepper motor.

4.2.5. Tapping wire control unit

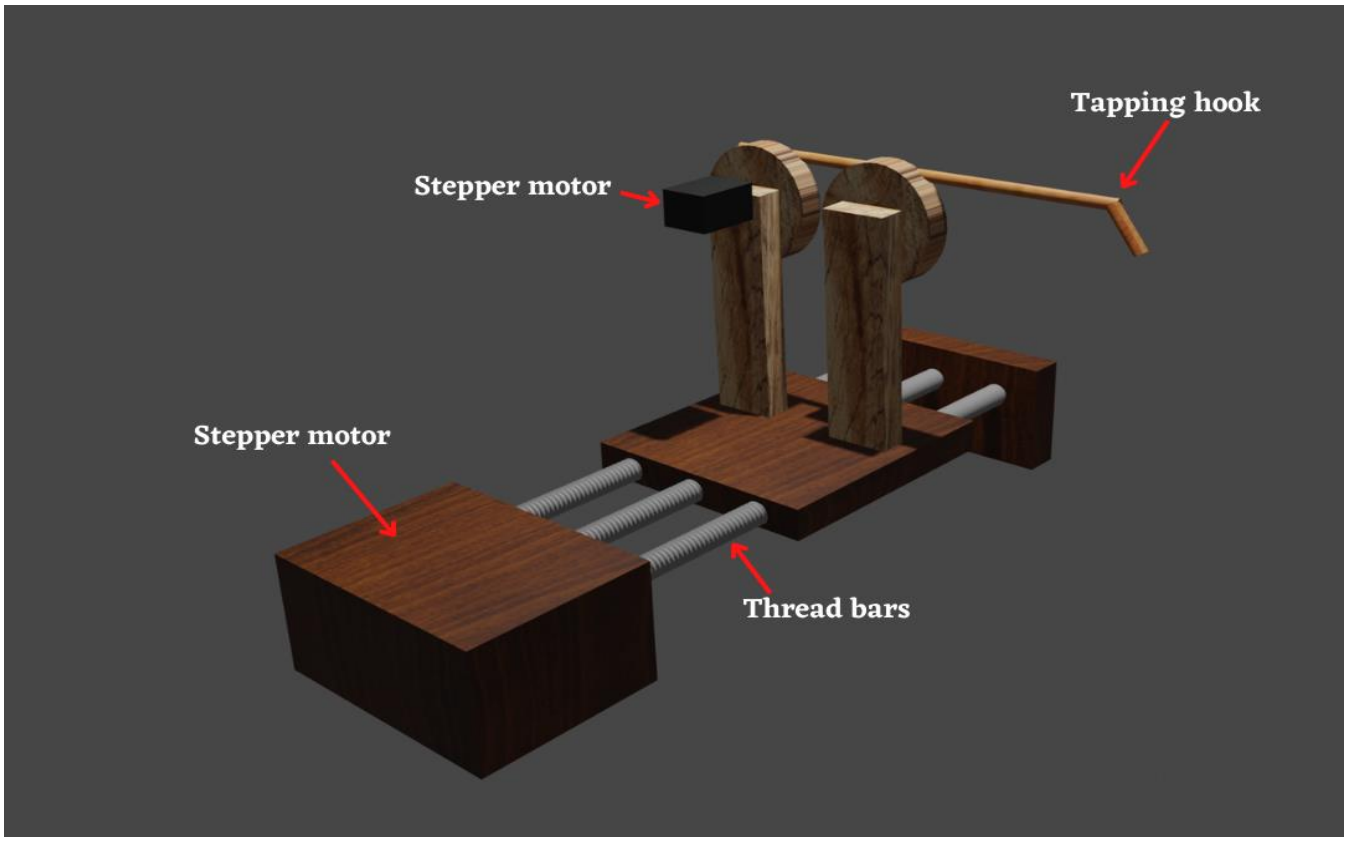

Figure 13 Tapping wire control unit

This unit is designed to operate only if the user inputs the tapping voltages to the system. While the secondary coil is winding when the tapping point comes, as the first task 4. The bobbin stops the rotation of the bobbin. Then the secondary arm moves and stops in front of the tapping wire control unit and it bends down to reach the tapping coil detainer(see *Figure 12*-).

Then the tapping control unit moves towards the bobbin and the which controls the hook also starts the rotation of the tapping hook. Then it folds the secondary coil to the hook and the tapping coil detainer in the 4. Bobbin rotation controlling unit (see *Figure 12- Bobbin rotation control unit*).

After completion of that process secondary arm moves to the previous location(Location where it stopped the winding process). Then the one taping arm bends and pastes the insulation tape on top of the coil(tapped coil).

After pasting tape 4. Bobbin rotation controlling unit restarts the rotation. While one of the taping arms, sticks the tape on the tapping coil part and the secondary arm controls the rest of the winding of the secondary coil and the other taping covers the winding coil on the bobbin. Finally, when the total winding process is over, the machine informs the user via a sound using the buzzer to eject the bobbin.

5. Testing and implementaiton

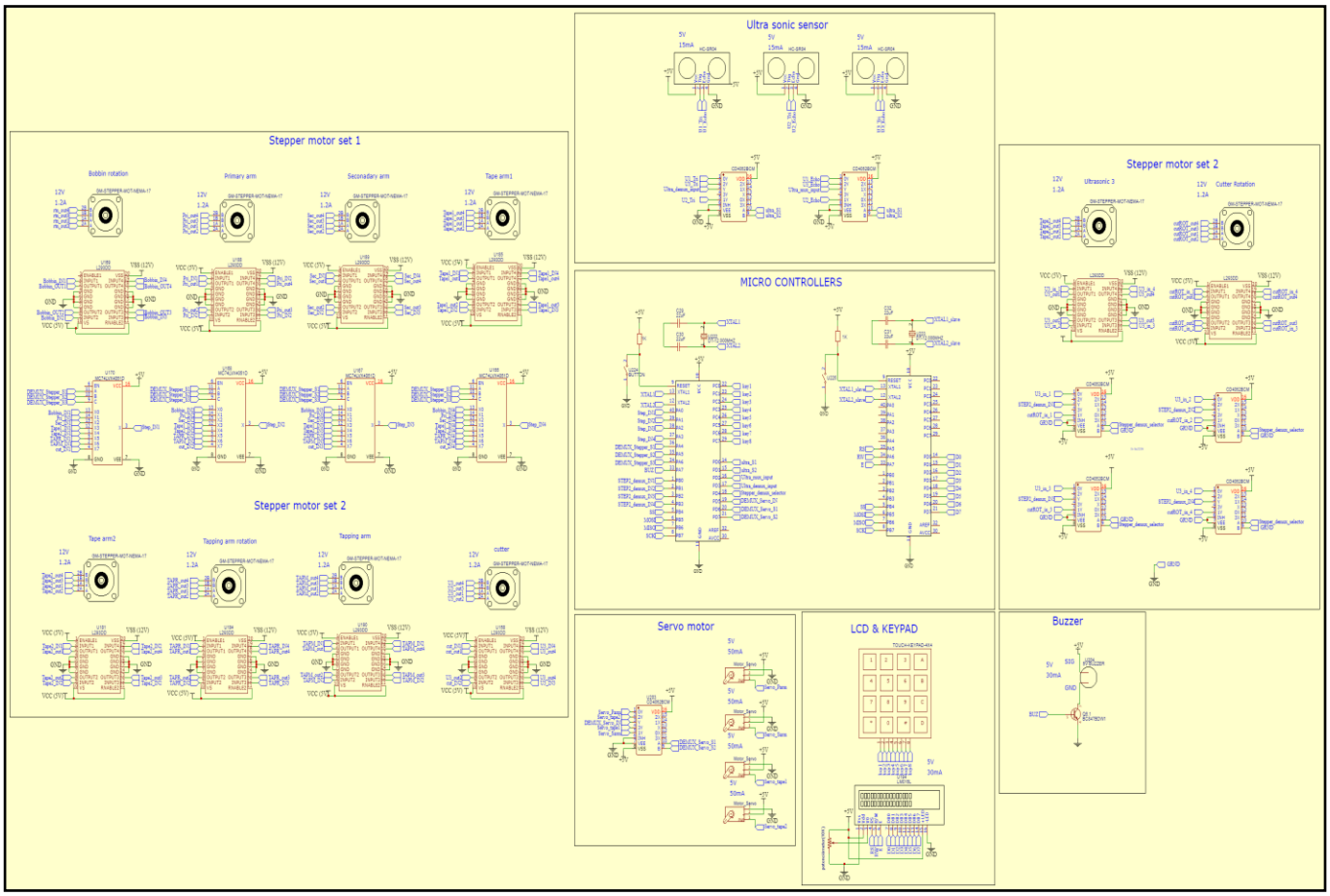

Figure 14

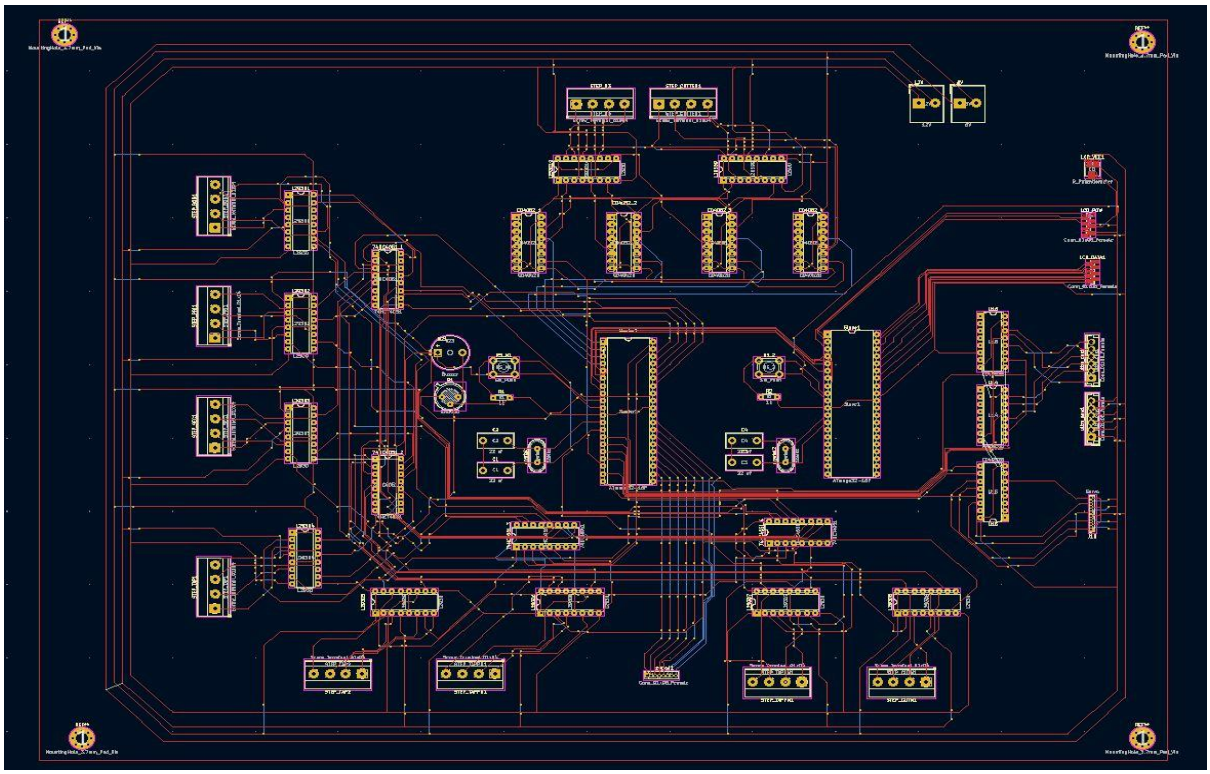

Figure 15

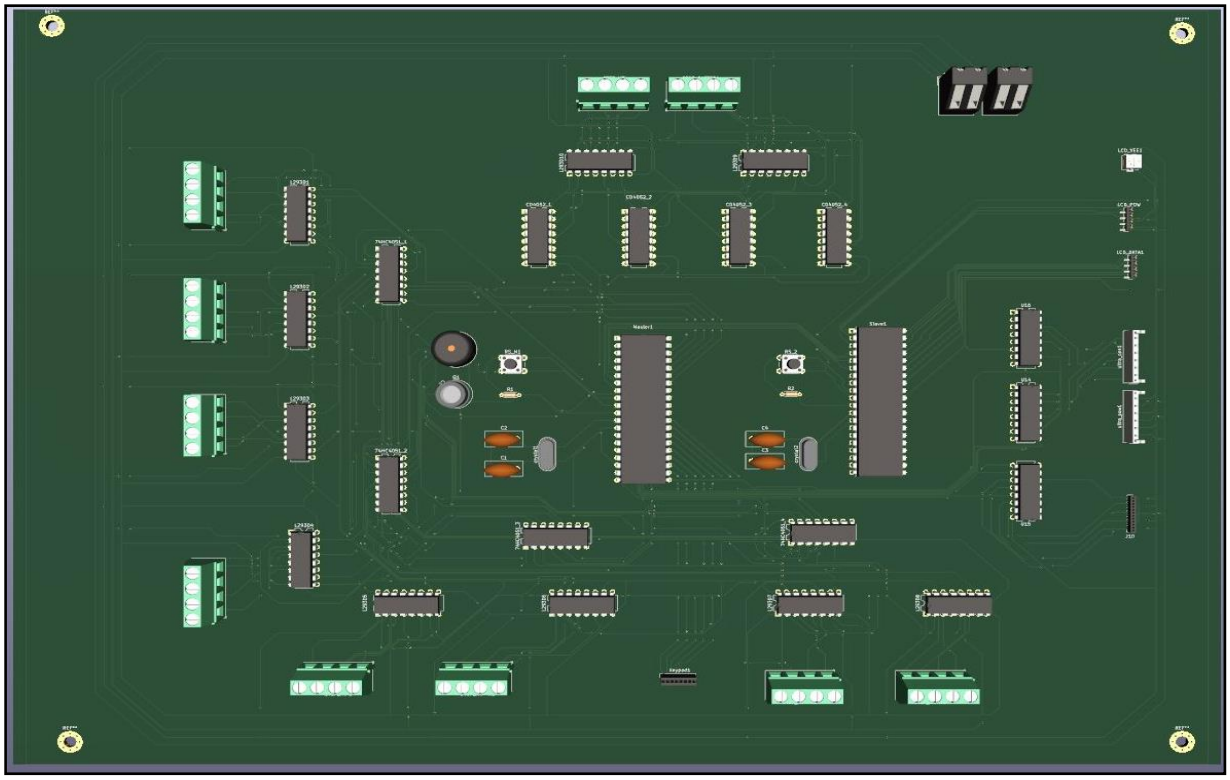

Figure 16

6. Further development

Insulation liquid unit

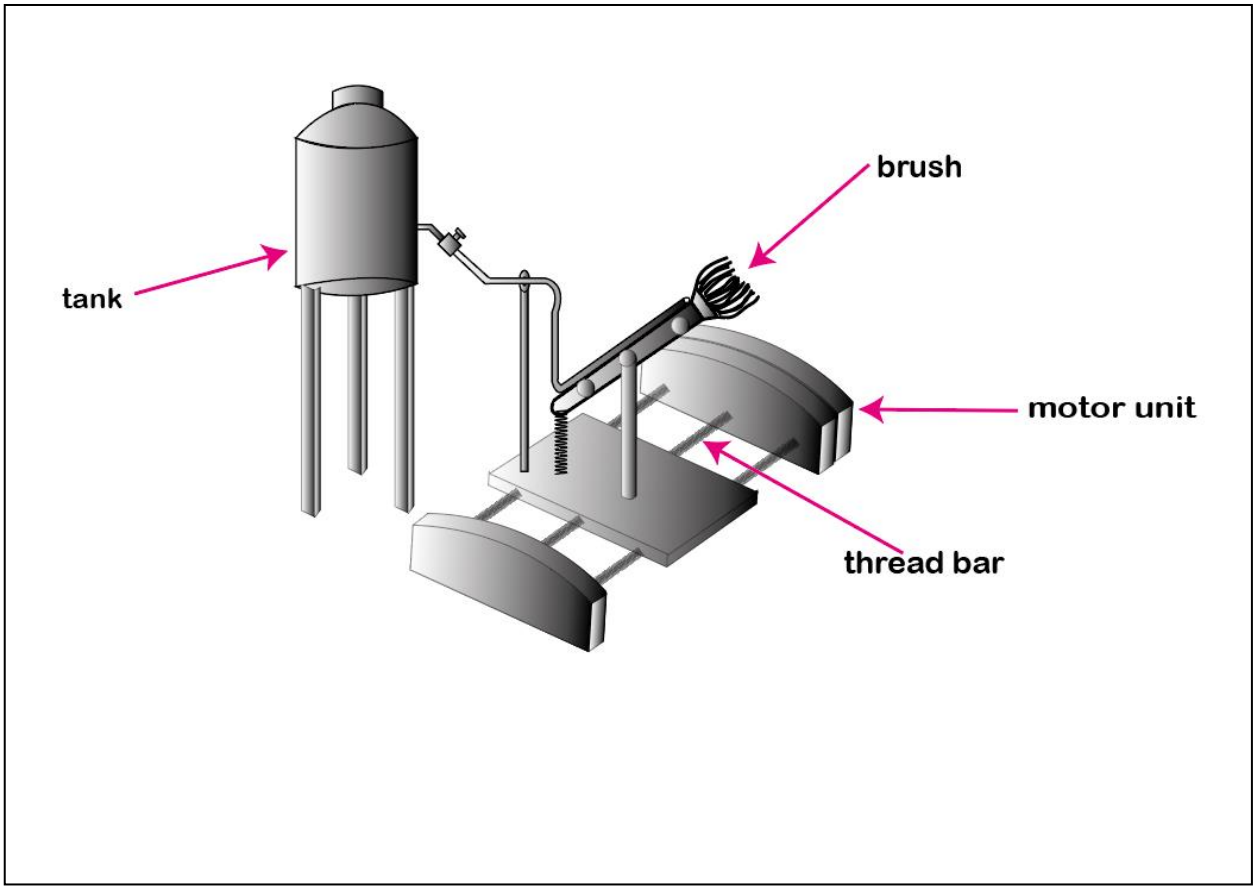

Figure 17- Insulation liquid unit

This separate unit is designed to apply the selak on the winding coil on the bobbin. With the help of the thread bars and iron bars, the brush can be moved horizontally parallel to the bobbin. In future upgrades our team hopes to insert in unit to the winder.

7. Estimated Cost

Item/sensor	quantity	Unit price	Cost
Ultrasonic sensor (HCSR04)	3	200.00	600.00
Stepper motor(Nema 17)	10	2700.00	27,000.00
Micro Servo motor(5V)	4	600.00	3000.00
Keypad	1	160.00	160.00
Cutter	1	500.00	500.00
16x2 LCD (LM016L)	1	1200.00	1200.00
CD4052BCM (1 to 4 demux)	7	400	2800.00
74HC4051D (1 to 8 demux)	4	500	2000.00
L293D DC motor driver	10	500.00	4500.00
Thread bar	10	200.00	2000.00
Power supply	1	2000.00	2000.00
Atmega 32	2	550.00	1100.00
Buzzer	1	100.00	100.00
Iron bar	16	200.00	3200.00
Wood	1	1500.00	1500.00
Total			51,660

Table 1-Cost Estimate

8. References

[1] F. BEDELL, “History of A-C Wave Form, Its Determination and Standardization,” *Trans. Am. Inst. Electr. Eng.*, vol. 61, no. 12, p. 864, 1942, DOI: 10.1109/T-AIEE.1942.5058456.

[2] J. Hindmarsh, “Electrical machines & their applications,” p. 662, 1984.

[3] “Ultrasonic Module HC-SR04 interfacing with AVR ATmega16/ATmega32 ...” https://www.electronicwings.com/avr-atmega/ultrasonic-module-hc-sr04-interfacing-with-atmega1632 (accessed Dec. 02, 2021).

[4] “Transformer Winding Machine, वाइंडिंग मशीन in Sidco Industrial Estate, Coimbatore, B.m.fabs | ID: 11170039212.” https://www.indiamart.com/proddetail/transformer-winding-machine-11170039212.html (accessed Dec. 02, 2021).

[5] “Transformer Winding Machine – Semi-Automatic Transformer Winding Machine For HV Coils Manufacturer from Bengaluru.” https://www.synthesiswinding.com/transformer-winding-machine.html (accessed Dec. 02, 2021).

[6] “Tap changer - Wikipedia.” https://en.wikipedia.org/w/index.php?title=Tap_changer&oldid=994260116 (accessed Dec. 04, 2021).

[7] “LCD16x2 Interfacing with AVR ATmega16/ATmega32 | AVR ATmega Contr...” https://www.electronicwings.com/avr-atmega/lcd16x2-interfacing-with-atmega16-32 (accessed Dec. 02, 2021).

[8] “L293D L293DD,” Accessed: Dec. 11, 2021. [Online]. Available: https://www.farnell.com/datasheets/1690387.pdf.

[9] “datasheet MC10174.” https://www.digchip.com/datasheets/parts/datasheet/343/MC10174-pdf.php (accessed Dec. 11, 2021).

[10] “4x4 Keypad interfacing with AVR ATmega16/ATmega32 | AVR ATmega Co...” https://www.electronicwings.com/avr-atmega/4x4-keypad-interfacing-with-atmega1632 (accessed Dec. 02, 2021).

Appendix A

Individuals' Contribution to the Project

Name of the student: Meegoda S.Y. (204129M)

1. Responsible part – 16*2 LCD , keypad, Winding parameters calculation ,PCB design, Saving essential data in EEPROM, power supply design

Learned to code atmega32 microcontroller using Atmel studio Software and learned to use Proteus software to simulate the circuit. I was aware of how to use the 16*2 LCD display and keypad. I have used the keypad in the master atmega micro controller, and I have used keypad in the slave atmega micro controller. I have established the communication between master and slave micro-controllers by using SPI(Serial peripheral interface). I have used 1024 KB eeprom, to save essential data in array data type which are used for winding parameters calculation part. And also I have designed the PCB design for the full machine and the power supply design by using the ki cad software. I drew the PCB design the 16*2 LCD display and keypad.

2. Specification

16*2 LCD display

I have configured the 16*2 LCD display in 8 bit mode and it's data pins are connected to the port C and it's command pins are connected to the port A in slave micro – controller. In the LCD there are 2 rows and 16 columns which is able to display maximum 32 characters and also each character in the LCD display is printed in 5*7 dot matrix. In the LCD there are 16 pins which are 8 data pins (D0-D7), 5 power pins (VSS,VDD,VEE,LEDA,LEDK) and 3 control pins (RS,RW,E).I have used the LCD display in the winder for display relevant information and parameters to the user.

4*4 keypad

I have used 4*4 keypad in order to take user inputs from the user. I have connected the keypad to the port C in master atmega micro-controller. In the keypad there 16 push buttons and 8 connection pins and it tolerate maximum 24V and 30mA. To pass the relevant pressed key value to the LCD display which is located in slave device, I have used SPI protocol. In the keypad I have used 0-9 digits to take integer inputs from the user and I have set '#' key as enter key.

3. Technique

1. After powering up the machine, user need to input primary voltage, primary voltage frequency, number of total amperages in the secondary, number of total output voltages, Relevant amperage and relevant voltages by using the keypad.

2. When user pressing a key in the keypad, the relevant pressed key is displayed on the LCD by using SPI protocol.

3. After taking the relevant inputs from the user, they are processed into a mathematical calculation.

4. In the calculation process primary coil standard wire gauge, secondary coil standard wire gauge, necessary core area for the bobbin, EI- lamination model number, number of turns for the necessary output voltages in the secondary coil,

5. After the calculation is over, primary coil standard wire gauge number, secondary coil wire gauge number, suitable core area(in cm^2) and relevant EI-Lamination model number is displayed to the user via the LCD display.

4. Schematic diagram

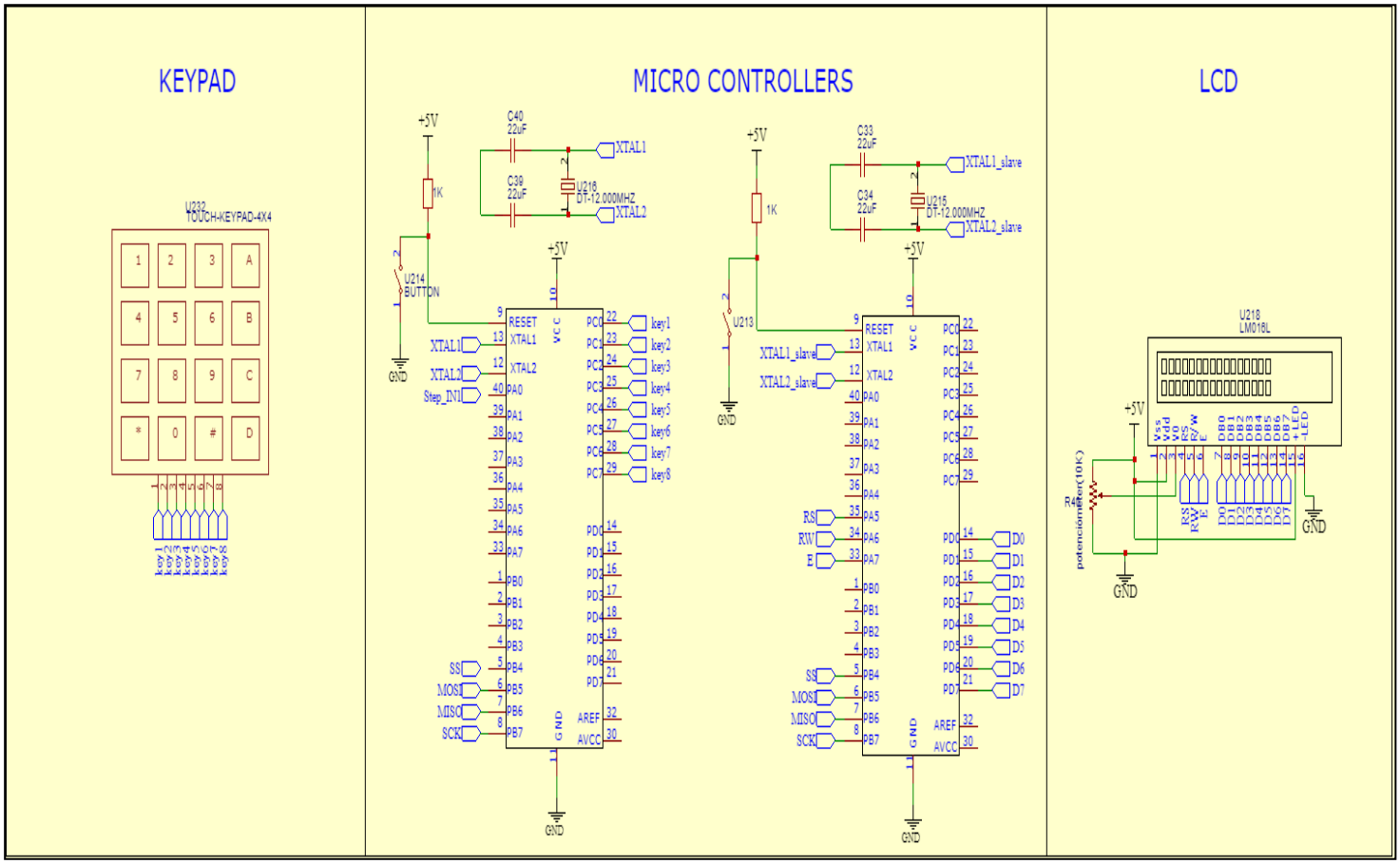

Figure 18

5. PCB design

16*2 LCD and 4*4 keypad

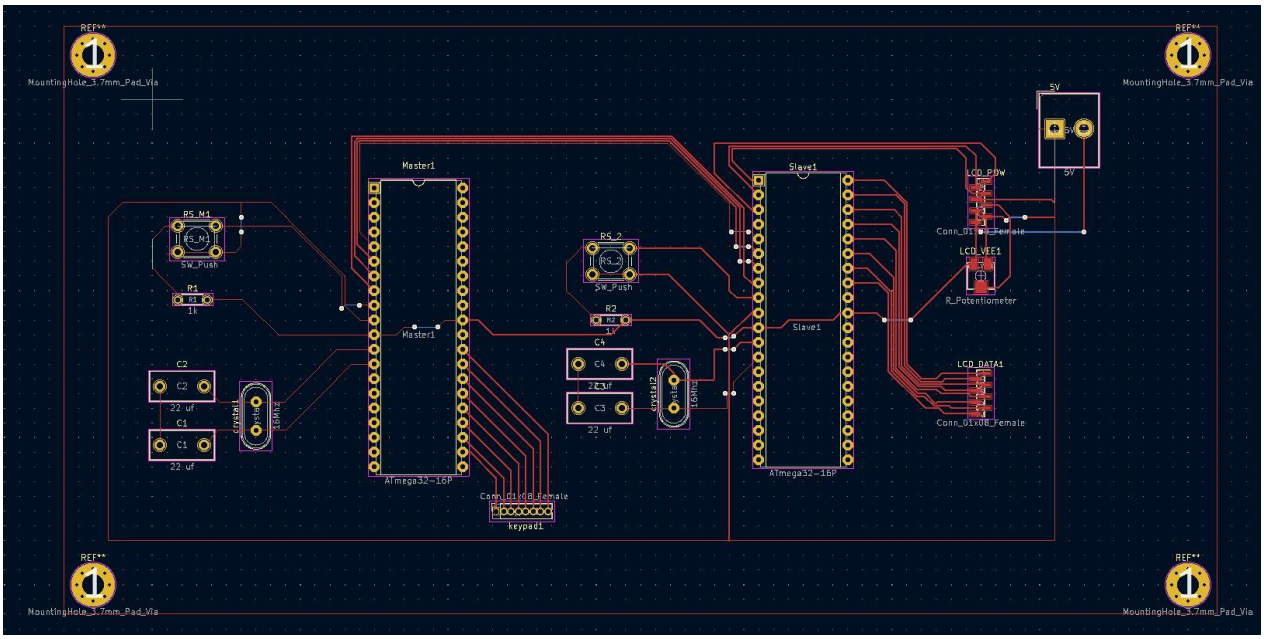

Figure 19

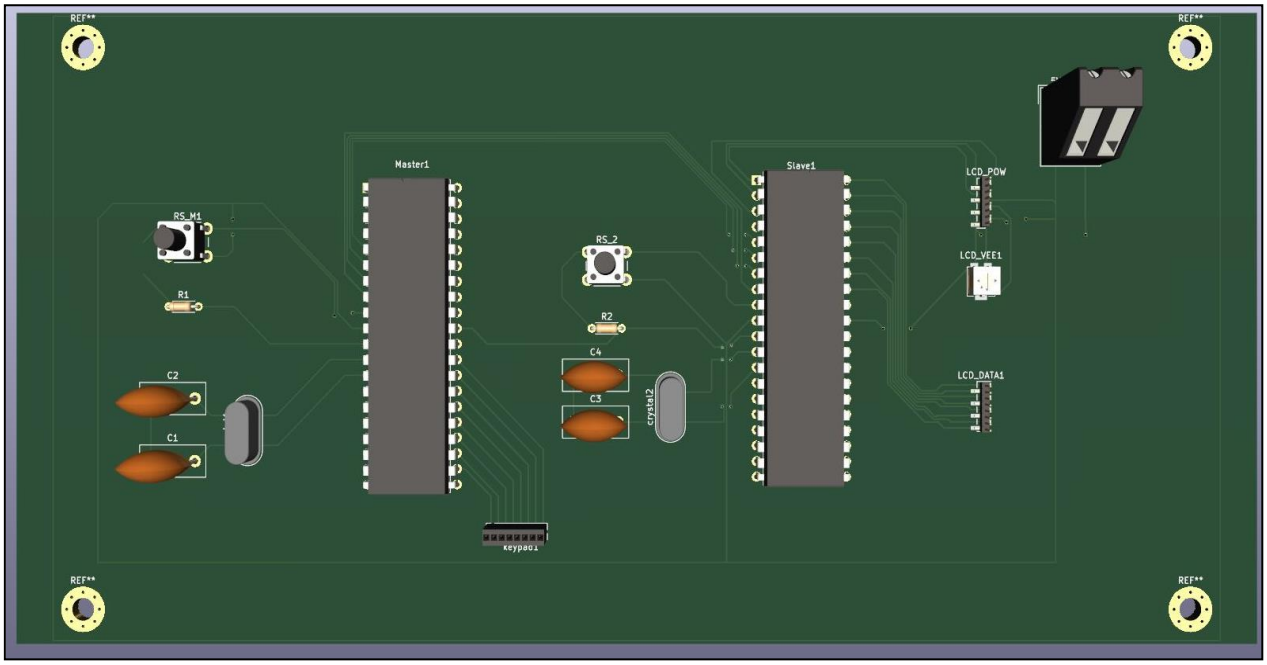

Figure 20

6. Power supply design

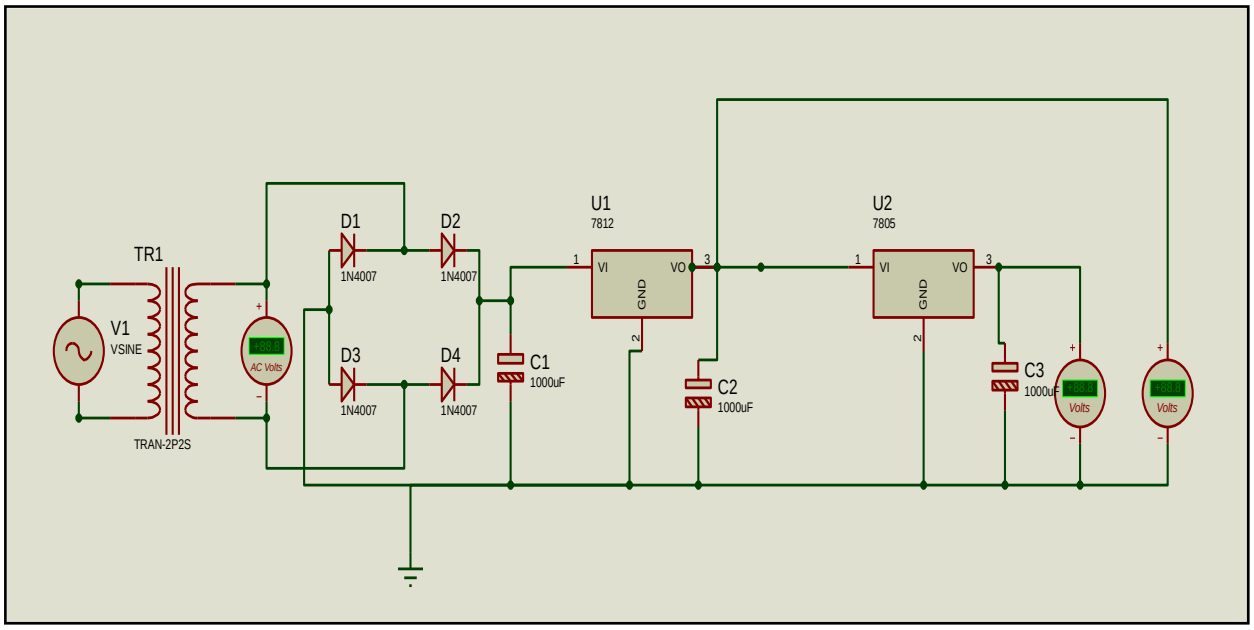

Figure 21

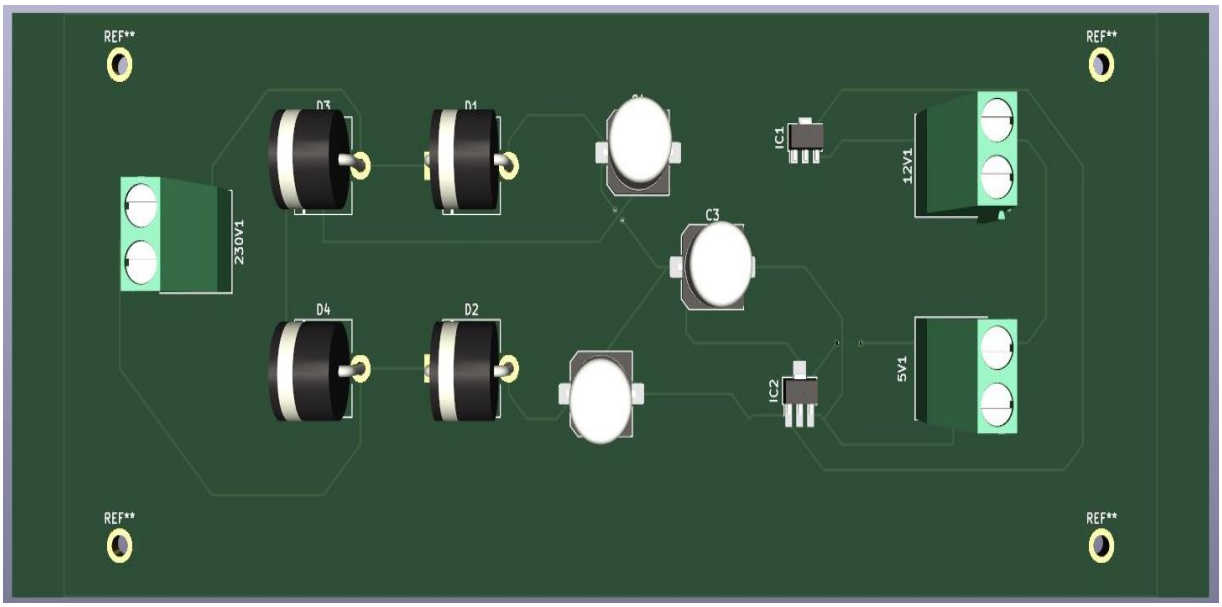

Figure 22

7. Proteus simulation

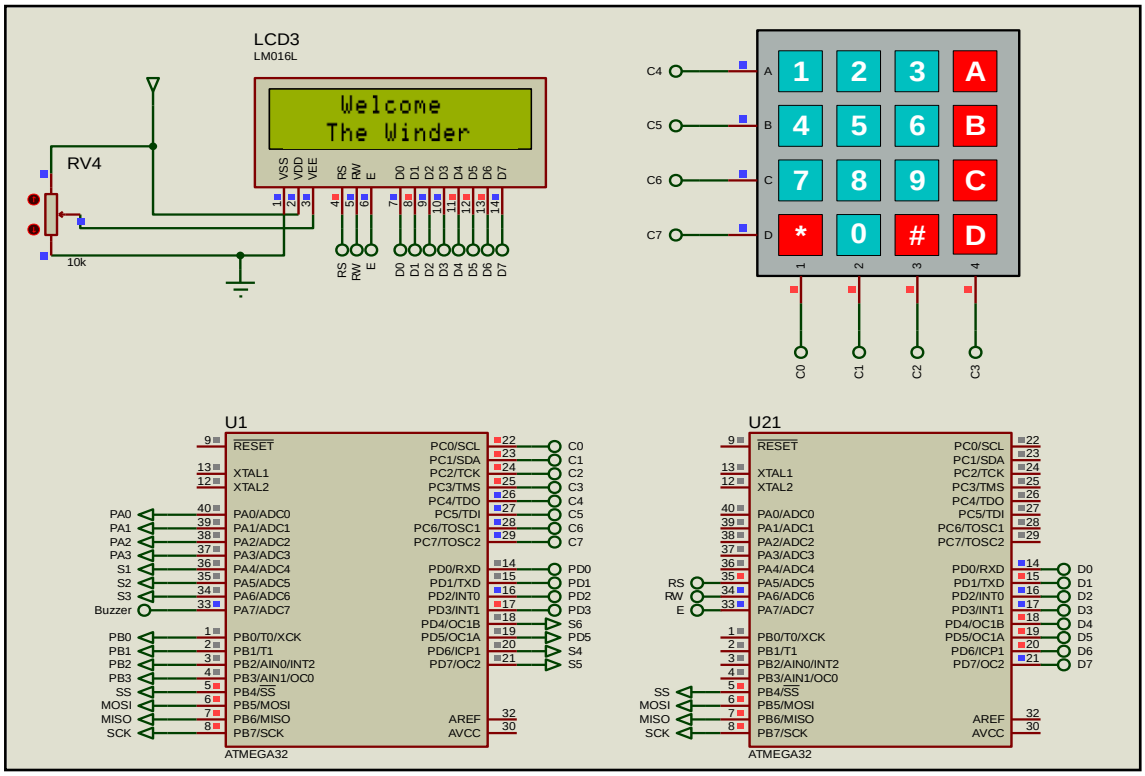

Figure 23

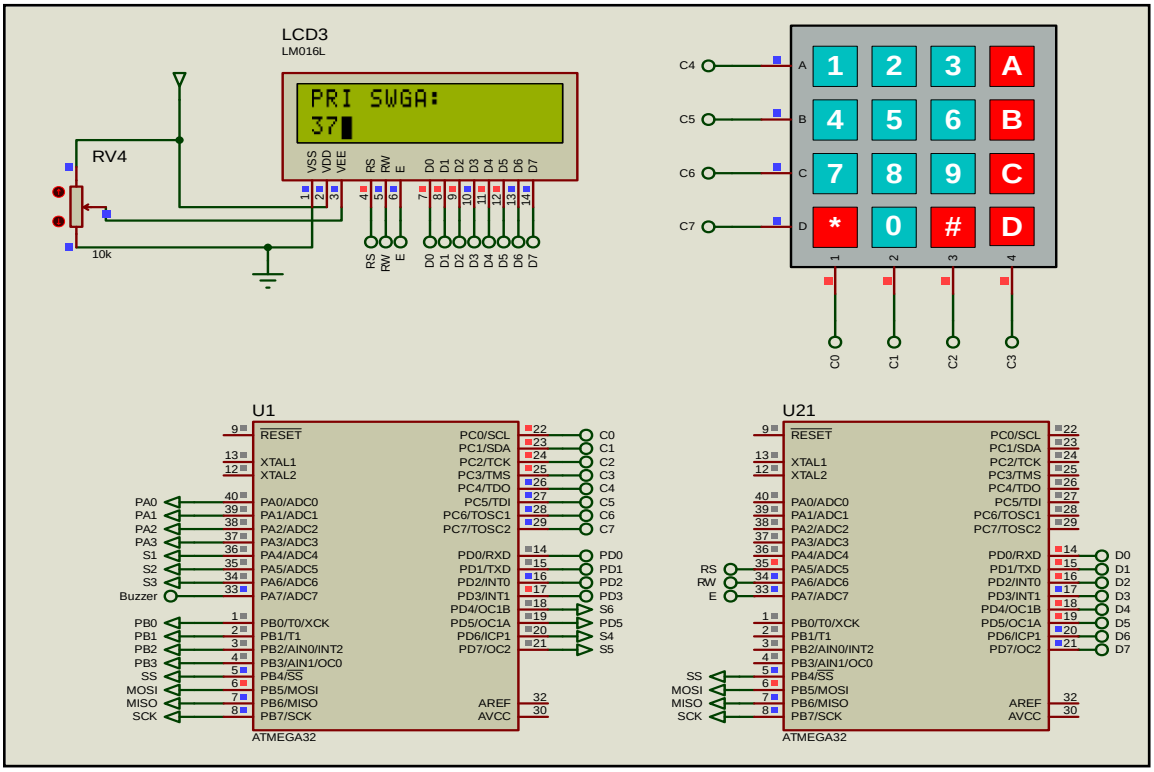

Figure 24

8. Code

Master atmega main.c

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include "keypad.h"
#include "eeprom_data.h"

float V_PRI,    //V_PRI=Primary voltage
F_PRI,          //F_PRI=Primary voltage frequency
POW_SEC,        //secondary power in MA
POW_SEC_TOT,    //total secondary power
POW_PRI,        //Primary power
C_AREA,         //core area
A_PRI,          //Priamry ampearage
Turns_V,        //Turns per volt
PRI_SWGA,       //Primary Standard wire gauge
PRI_TSQR,       //Primary wire turns per square centimetres
TOT_PRI_AREA=0,     //Total area taken by primary coil
TOT_SEC_AREA=0,     //Total area taken by secondary coil
TOT_AREA=0,         //Total area taken by both primary and secondary coil
D_PRI               //diameter of the primary coil
;
int ACOUNT=1,       //number of amps in sec
VCOUNT=0,           //number of volts in sec
```

```
SEC_VTOT=1,          //secondary total number of voltages
NOT_PRI,          //number of turns in primary
P_SWGA,                              //primary SWGA location
EI_LAMIN      //EI_LAM = EI laminations model number
;
Int Z_length;          //ultra-sonic length 3

float RAM_SWGA[34]={47,46,45,44,43,42,41,40,39,38,37,36,35,34,33,
32,31,30,29,28,27,26,25,24,23,22,21,20,19,18,17,16,15,14};                    //Standard wire gauge

float RAM_AMP[34]={8,10,13,16,19,24,30,36,39,50,72,90,110,
        132,151,182,221,240,290,344,422,508,628,760,940,1210,1600,2030,2570,3600,
4900,6430,8410,                  //maximum ampearage in coil
10040};

float
RAM_TURNS[34]={27546,20223,14392,11457,9337,7755,6543,5595,4838,3507,2800,2286,19
02,                //Turns per square centimetres
1608,1308,1137,997,881,711,609,504,415,341,286,242,176,137,106,87.4,60.8,45.4,35.2,26.8,21.
5};

char RAM_EILAM[30][4]={"17","12A","74","23","30","21","31","10","15","33","1",
"14","11","34","3","9","9A","11A","4A","2","16",   //EI lamination model number
"5", "4AX","13","75","4","7","6","35A","8"};

float RAM_CORE[30]=
{1.213,1.897,2.284,2.723,3.000,3.329,3.703,4.439,4.839,5.880,6.555,6.555,7.259,7.259,
        //Core area of bobbin
        7.562,7.865,7.865,9.072,10.284,10.891,10.891,12.704,13.039,14.117,15.324,
15.865,18.969,19.356,39.316,40.803};
```

```
float
RAM_DIA[34]={0.05,0.06,0.07,0.08,0.09,0.10,0.11,0.12,0.13,0.15,0.17,0.19,0.21,0.23,0.25,0.2
7,                              //diameter of coils
0.29,0.31,0.34,0.37,0.41,0.45,0.50,0.55,0.61,0.71,0.81,0.91,1.01,1.21,1.42,1.62,1.82,2.03};

float M_SWGA[34];
float M_MAMP[34];
float M_TPCS[34];
float vCore_area[30];
char M_EI_LAM[30][4];
float M_EEP_DIA[34];

int main(void)
{
  write_eeprom_array(SWGA,RAM_SWGA,sizeof(SWGA));
  write_eeprom_array(MAMP,RAM_AMP,sizeof(MAMP));
  write_eeprom_array(TPCS,RAM_TURNS,sizeof(TPCS));
  write_eeprom_array(Core_area,RAM_CORE,sizeof(Core_area));
  write_eeprom_array(EI_LAM,RAM_EILAM,sizeof(EI_LAM));
  write_eeprom_array(EEP_DIA,RAM_DIA,sizeof(EEP_DIA));


read_eeprom_array(M_SWGA,SWGA,sizeof(M_SWGA));
read_eeprom_array(M_MAMP,MAMP,sizeof(M_MAMP));
read_eeprom_array(M_TPCS,TPCS,sizeof(M_TPCS));
read_eeprom_array(vCore_area,Core_area,sizeof(vCore_area));
read_eeprom_array(M_EI_LAM,EI_LAM,sizeof(M_EI_LAM));
read_eeprom_array(M_EEP_DIA,EEP_DIA,sizeof(M_EEP_DIA));

        key_enter();
        int num1=0;
```

```
//priamary voltage
num1 = key_in();//converts string to integer
V_PRI=(float) num1;
//Primary frequency
num1 = key_in();
F_PRI=(float) num1;
//number of ampearages
num1 = key_in();
ACOUNT=num1;
pass_int(ACOUNT);
float A_SEC[ACOUNT];        //secondary ampearage array
//total number of voltages
num1 = key_in();
SEC_VTOT=num1;

int NO_OF_VOLTS[ACOUNT];        //Number of voltages for each ampearage
float V_SEC[SEC_VTOT];          //secondary total voltages array
int No_Tu_SEC[SEC_VTOT];        //secondary total no of turns array
float SEC_SWGA[ACOUNT];           //Secondary standard wire gauge
float SEC_TSQR[ACOUNT];           //Secondary turns per square centimetres
float D_SEC[ACOUNT];              //diameter array of secondary coil
int final_turns_array[SEC_VTOT];      //final number of turns for winding
int final[SEC_VTOT];

int a=0;
for(int i=0;i<ACOUNT;i++)
{

        //assigning amearage value
        num1 = key_in();
        A_SEC[i]=(float)num1;
```

```
        //assigning number of voltages to relevant ampearage
        num1 = key_in();
        VCOUNT=num1;

        NO_OF_VOLTS[i]=VCOUNT;
        pass_int(VCOUNT);

        for(int j=0;j<VCOUNT;j++)
        {
                num1 = key_in();
                V_SEC[a]=(float)num1;
                a++;
        }

}

//calculating secondary total power
int temp=0;
for (int i = 0; i <ACOUNT ; i++)
{
        for(int j=0; j<NO_OF_VOLTS[i] ; j++)
        {
                POW_SEC =POW_SEC + (A_SEC[i]*V_SEC[temp]);
                temp++;
        }
}
//total secondary power calculation
POW_SEC_TOT=POW_SEC/1000;

//Primary power calculation
```

```
POW_PRI = POW_SEC_TOT * 1.15;  //0.15=energy loss

//finding core area(Metric standad calculation)
C_AREA = 1.152 * sqrt(POW_PRI); //unit is square centimetres

//Finding primary ampearage
A_PRI = (POW_PRI / V_PRI) * 1000; //I=P/V ampearage is in mA

//Finding turns per volt
Turns_V = 10000 / (4.4 * F_PRI * C_AREA * 1.3); //1.3= Magnetic flux

//finding the turns per volt in Primary winding and secondary winding
float PT = Turns_V * V_PRI;
NOT_PRI = (int) PT + 1;

//finding turns per volt in secondary
for (int i = 0; i <SEC_VTOT ; i++)
{
        float ST = Turns_V*V_SEC[i];
        No_Tu_SEC[i]=(int) ST + 1;
}
//selecting relevant primary ampearages
for (int i= 0; i <34; i++)
{
        if (RAM_AMP[i] >= A_PRI)
        {
                PRI_SWGA = RAM_SWGA[i];
                P_SWGA=i;
                break;
        }
        else if ((RAM_AMP[i] < A_PRI) && (RAM_AMP[i + 1] > A_PRI))
```

```
        {

                PRI_SWGA = RAM_SWGA[i + 1];
                P_SWGA=(i+1);
                break;
        }
}
//selecting the relevant ampearage and SWGA for secondary
for (int j = 0; j < ACOUNT; j++)
{
        for (int i = 0; i <= 34; i++)
        {
                if (RAM_AMP[i] >= A_SEC[j])
                {

                        SEC_SWGA[j] = RAM_SWGA[i];
                        break;
                }
                else if((RAM_AMP[i] < A_SEC[j]) && (RAM_AMP[i + 1] > A_SEC[j]))
                {
                        SEC_SWGA[j] = RAM_SWGA[i + 1];
                        break;
                }
        }
}

//selected diameter array for secondary coils
for (int j = 0; j < ACOUNT; j++)
{
        for (int i = 0; i <= 34; i++)
        {
```

```
                if (RAM_SWGA[i] == PRI_SWGA)
                {
                        D_SEC[j] = RAM_DIA[i];
                }
        }
}

//Finding primary wire square centimetres
for (int i = 0; i < 34; i++)
{
        if (RAM_SWGA[i] == PRI_SWGA)
        {
                PRI_TSQR =RAM_TURNS[i];
                break;
        }
}

//Finding primary wire diameter
for (int i = 0; i < 34; i++)
{
        if (RAM_SWGA[i] == PRI_SWGA)
        {
                D_PRI = RAM_DIA[i];
                break;
        }

}
//Finding secondary turns per square centimetres
for (int j = 0; j < ACOUNT; j++)
{
```

```
        for (int i = 0; i <= 34; i++)
        {
                if (RAM_SWGA[i] == SEC_SWGA[j])
                {
                        SEC_TSQR[j] = RAM_TURNS[i];
                        break;
                }
        }
}

//calculating total area taken by the primary coil
TOT_PRI_AREA = TOT_PRI_AREA + (NOT_PRI / PRI_TSQR);

//secondary coil total area
int d=0;
float temp_SQR=0;
for(int i=0;i<ACOUNT;i++)
{
        for (int j = 0; j < NO_OF_VOLTS[i]; j++)
        {
                temp_SQR=No_Tu_SEC[d]/SEC_TSQR[i];
                TOT_SEC_AREA=TOT_SEC_AREA+temp_SQR;
                d++;
        }

}
//calculating total area taken by the both primary and secondary coil
TOT_AREA = TOT_PRI_AREA + TOT_SEC_AREA;
//selecting relevant EI lamination
for (int i = 0; i < 30; i++)
{
```

```
            if (RAM_CORE[i] >= TOT_AREA)
            {
                  EI_LAMIN =i;
                  break;
            }
            else if ((RAM_CORE[i] < TOT_AREA) && (RAM_CORE[i + 1] >
TOT_AREA))
            {
                  EI_LAMIN=(i +1);
                  break;
            }
      }
      //secondary final number of turns array
      int f=0;
      for(int i=0;i<ACOUNT;i++)
      {
            for (int j = 0; j < NO_OF_VOLTS[i]; j++)
            {
                  if(j==0)
                  {
                        final_turns_array[f] = No_Tu_SEC[f];
                  }
                  else
                  {
                        final_turns_array[f] = No_Tu_SEC[f] - No_Tu_SEC[f-1];
                  }
                  f++;
            }
      }

      //passing relevant values to LCD using SPI
```

```
        //printing primary SWGA
        pass_int(P_SWGA);
        key_enter();
        //printing secondary SWGA
        int S_VAL;
        for(int i=0;i<ACOUNT;i++)
        {

                S_VAL=SEC_SWGA[i];
                _delay_ms(500);
                pass_int(S_VAL);
                _delay_ms(1000);
                key_enter();
                _delay_ms(500);

        }

        //printing core area
        int CA;
        CA=(int) C_AREA;
        pass_int(CA);
        key_enter();

        //priting ei lamination
        pass_int(EI_LAMIN);
        key_enter();

//Place the bobbin
        buzzer();
        key_enter();
```

```
//Measuring x length
x1_distance=ultrasonic_1();
x2_distance=ultrasonic_2();
x_measurement=x_fix-x1_distance-x2_distance;

//rotate the bobbin
buzzer();
key_enter();

//Measuring y length
y1_distance=ultrasonic_1();
y2_distance=ultrasonic_2();
y_measurement=y_fix-y1_distance-y2_distance;

//start measuring z length
key_enter();

//Measuring Z length
z1_distance=ultrasonic_3();
z_measurement=z_fix-z1_distance;



//Bobbin Measuring Ending
cal_core= x_measurement*y_measurement;

if(cal_core>CA)
{
        int pass=1;
        pass_int(pass);
        key_enter();
}
```

```
else
{
        int fail=2;
        pass_int(fail);
        key_enter();
}
key_enter();

//press # to start winding
key_enter();

int primary_side= z_measurement/(D_PRI*0.01);
primary_winding(D_PRI,NOT_PRI,primary_side);

int q=0;
int temp1=ACOUNT-1;
if(ACOUNT>1)
{
        for(int i=0;i<ACOUNT;i++)
        {
                int lenth=NO_OF_VOLTS[i];
                _delay_ms(500);
                int final[lenth];
                for (int j = 0; j < lenth; j++)
                {
                        final[j]=final_turns_array[q];
                        _delay_ms(1000);
                        q++;
                }
                for (int j = 0; j < NO_OF_VOLTS[i]; j++)
                {
```

```
                        int secondary_side= z_measurement/D_SEC[i]*0.01;

secondary_winding(D_SEC[i],secondary_side,final,NO_OF_VOLTS[i]);
                }
                if(i==temp1)
                {
                        int temp2=30;
                        pass_int(temp2);
                        buzzer();
                        key_enter();
                }
                else
                {
                        int temp2=10;
                        pass_int(temp2);
                        buzzer();
                        key_enter();
                }
        }
}

else
{
        int a=0;
        int final[NO_OF_VOLTS[0]];
        for (int j = 0; j < NO_OF_VOLTS[0]; j++)
        {
                final[j]=final_turns_array[a];
                a++;
        }
        int secondary_side= z_measurement/(D_SEC[0]*0.01);
```

```
                secondary_winding_no_tap(D_SEC[0],final[0],secondary_side);
                buzzer();
                int temp3=20;
                pass_int(temp3);
                key_enter();

        }
}
```

Keypad.h

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

#define MOSI 5                  /* Define SPI bus pins */
#define MISO 6
#define SCK 7
#define SS 4
#define SS_Enable PORTB &= ~(1<<SS)                     /* Define Slave enable */
#define SS_Disable PORTB |= (1<<SS)                     /* Define Slave disable */

#define KEY_PRT PORTC
#define KEY_DDR    DDRC
#define KEY_PIN     PINC

void SPI_Init()                 /* SPI Initialize function */
{
        DDRB |= (1<<MOSI)|(1<<SCK)|(1<<SS);             /* Make MOSI, SCK, 0th pin
direction as output pins */
        DDRB &= ~(1<<MISO);                             /* Make MISO pin as input pin */
```

```
	PORTB |= (1<<SS);			/* Disable slave initially by making high on SS pin
*/
	SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0); /* Enable SPI, Enable in master mode, with
Fosc/16 SCK frequency */
	SPSR &= ~(1<<SPI2X);					/* Disable speed doubler */
}

void SPI_Write(char data)		/* SPI write data function */
{
	//char flush_buffer;
	SPDR = data;				/* Write data to SPI data register */
	while(!(SPSR & (1<<SPIF)));		/* Wait till transmission complete */
	//flush_buffer = SPDR;			/* Flush received data */
	/* Note: SPIF flag is cleared by first reading SPSR (with SPIF set) and then accessing
SPDR hence flush buffer used here to access SPDR after SPSR read */
}
unsigned char keypad[4][4] = {{'1','2','3','A'},
{'4','5','6','B'},
{'7','8','9','C'},
{'*','0','#','D'}};

unsigned char colloc, rowloc;

char keyfind()
{
	while(1)
	{
		KEY_DDR = 0xF0;		/* set port direction as input-output */
		KEY_PRT = 0xFF;

		do
```

```
{
        KEY_PRT &= 0x0F;      /* mask PORT for column read only */
        asm("NOP");
        colloc = (KEY_PIN & 0x0F); /* read status of column */
}while(colloc != 0x0F);

do
{
        do
        {
                _delay_ms(20);            /* 20ms key debounce time */
                colloc = (KEY_PIN & 0x0F); /* read status of column */
        }
        while(colloc == 0x0F);        /* check for any key press */

        _delay_ms (40);                   /* 20 ms key debounce time */
        colloc = (KEY_PIN & 0x0F);

}
while(colloc == 0x0F);

/* now check for rows */
KEY_PRT = 0xEF;           /* check for pressed key in 1st row */
asm("NOP");
colloc = (KEY_PIN & 0x0F);
if(colloc != 0x0F)
{
        rowloc = 0;
        break;
}
```

```
        KEY_PRT = 0xDF;             /* check for pressed key in 2nd row */
        asm("NOP");
        colloc = (KEY_PIN & 0x0F);
        if(colloc != 0x0F)
        {
                rowloc = 1;
                break;
        }

        KEY_PRT = 0xBF;             /* check for pressed key in 3rd row */
        asm("NOP");
        colloc = (KEY_PIN & 0x0F);
        if(colloc != 0x0F)
        {
                rowloc = 2;
                break;
        }

        KEY_PRT = 0x7F;             /* check for pressed key in 4th row */
        asm("NOP");
        colloc = (KEY_PIN & 0x0F);
        if(colloc != 0x0F)
        {
                rowloc = 3;
                break;
        }
}

if(colloc == 0x0E)
return(keypad[rowloc][0]);
else if(colloc == 0x0D)
```

```
        return(keypad[rowloc][1]);
        else if(colloc == 0x0B)
        return(keypad[rowloc][2]);
        else
        return(keypad[rowloc][3]);
}
void key_enter()        //enter key function
{
        char key;
        while(1)
        {
                key=keyfind();
                if(key=='#')
                {
                        SPI_Init();
                        SS_Enable;
                        SPI_Write(key);
                        break;
                }
        }
}
int key_in()            //key input taking function
{
        char temp[5]="0";
        int n=0;
        char key;
        while(1)
        {

                key=keyfind();
                if(key=='#')
```

```
                {
                        SPI_Init();
                        SS_Enable;
                        SPI_Write(key);//writing pressed key to slave
                        break;
                }
                else
                {

                        SPI_Init();
                        SS_Enable;
                        SPI_Write(key);
                        temp[n]=key;
                        n++;
                }


        }
        int num = atoi(temp);
        return num;
}
void pass_int(int x)            //Sending integer to slave
{
        int num;
        num=x;

        SPI_Init();
        SS_Enable;
        SPI_Write(num);

}
```

eeprom_data.h

```
#include <avr/io.h>
#include <avr/eeprom.h>

// macro for easier usage
#define read_eeprom_array(address,value_p,length) eeprom_read_block ((void *)value_p, (const
void *)address, length)
#define write_eeprom_array(address,value_p,length) eeprom_write_block ((const void
*)value_p, (void *)address, length)

//declare an eeprom array
float EEMEM SWGA[34]; //standard wire gauge
float EEMEM MAMP[34]; //Maximum ampearage
float EEMEM TPCS[34]; //Turns per square centimetre
float EEMEM Core_area[30]; //Core area
char EEMEM EI_LAM[30][4]; //EI-lamination model number
float EEMEM EEP_DIA[34]; //Diametre of the coil

// declare a ram array and initialize
float RAM_SWGA[34]={47,46,45,44,43,42,41,40,39,38,37,36,35,34,33,
32,31,30,29,28,27,26,25,24,23,22,21,20,19,18,17,16,15,14}; //Standard wire gauge

float RAM_AMP[34]={8,10,13,16,19,24,30,36,39,50,72,90,110,
132,151,182,221,240,290,344,422,508,628,760,940,1210,1600,2030,2570,3600, 4900,6430,8410,
//maximum ampearage in coil
10040};

float
RAM_TURNS[34]={27546,20223,14392,11457,9337,7755,6543,5595,4838,3507,2800,2286,19
02, //Turns per square centimetres
```

```
1608,1308,1137,997,881,711,609,504,415,341,286,242,176,137,106,87.4,60.8,45.4,35.2,26.8,21.
5};

char RAM_EILAM[30][4]={"17","12A","74","23","30","21","31","10","15","33","1",
"14","11","34","3","9","9A","11A","4A","2","16", //EI lamination model number
"5", "4AX","13","75","4","7","6","35A","8"};

float RAM_CORE[30]=
{1.213,1.897,2.284,2.723,3.000,3.329,3.703,4.439,4.839,5.880,6.555,6.555,7.259,7.259, //Core
area of bobbin
7.562,7.865,7.865,9.072,10.284,10.891,10.891,12.704,13.039,14.117,15.324,
15.865,18.969,19.356,39.316,40.803};

float
RAM_DIA[34]={0.05,0.06,0.07,0.08,0.09,0.10,0.11,0.12,0.13,0.15,0.17,0.19,0.21,0.23,0.25,0.2
7, //diameter of coils
0.29,0.31,0.34,0.37,0.41,0.45,0.50,0.55,0.61,0.71,0.81,0.91,1.01,1.21,1.42,1.62,1.82,2.03};
```

Slave main.c

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>

#include "slave_SPI.h"
```

```
#include "LCD_8bit.h"

int
ACT,            //total no of ampearages
VCT,            //Relevant ampearage no of voltages
P_SWGA,             //Primary SWGA array location
S_SWGA,             //Secondary SWGA
C_AR,           //Core area
EI_LA,              //EI lamination array location
Bob_result,     //Bobbin measuring unit result
win_result       //next coil insertion result
;

//data arrays
int RAM_SWGA[34]={47,46,45,44,43,42,41,40,39,38,37,36,35,34,33,
32,31,30,29,28,27,26,25,24,23,22,21,20,19,18,17,16,15,14};                  //Standard wire gauge

char RAM_EILAM[30][4]={"17","12A","74","23","30","21","31","10","15","33","1",
"14","11","34","3","9","9A","11A","4A","2","16",                    //EI lamination model
number
"5", "4AX","13","75","4","7","6","35A","8"};
//enter key receiving function
void key_enter_receive()
{

        char key1='0';

        while(1)
        {

                SPI_Init();
```

```
                key1=SPI_Receive();
                if(key1=='#')
                {

                        LCD_Clear();
                        _delay_ms(500);
                        break;
                }


        }


}
//Enter key receiving function
void key_receive()
{
        char value;
        while(1)
        {
                SPI_Init();
                value = SPI_Receive();//Receiveing the pressed key from master
                if(value=='#')
                {
                        LCD_Clear();
                        _delay_ms(500);
                        break;
                }
                else
                {
                        LCD_Char(value);//printing pressed key in LCD
```

```
                }
        }

}

int main(void)
{
        LCD_Init();
        LCD_String("    Welcome");
        LCD_Command(0xC0); //setting cursor to second line
        _delay_ms(300);
        LCD_String("   The Winder");
        key_enter_receive();

        LCD_String("Enter primary vo");
        LCD_Command(0xC0);
        LCD_String("ltage :");
        LCD_Command(0x0F);
        key_receive();

        LCD_String("Enter primary fr");
        LCD_Command(0xC0);
        LCD_String("equency :");
        _delay_ms(300);
        LCD_Command(0x0F);
        key_receive();

        LCD_String("Enter NO of ");
        LCD_Command(0xC0);
        LCD_String("ampearages:");
        _delay_ms(300);
```

```
LCD_Command(0x0F);
key_receive();

ACT=get_value();

LCD_String("Enter NO of ");
LCD_Command(0xC0);
LCD_String("total volts:");
_delay_ms(300);
LCD_Command(0x0F);
key_receive();
for(int i=0;i<ACT;i++)
{
        LCD_String("Enter ampearage");
        LCD_Command(0xC0);
        LCD_String("(mA):");
        _delay_ms(300);
        LCD_Command(0x0F);
        key_receive();

        LCD_String("Enter no of");
        LCD_Command(0xC0);
        LCD_String("volts:");
        _delay_ms(300);
        LCD_Command(0x0F);
        key_receive();

        VCT=get_value();
        for(int j=0;j<VCT;j++)
        {
                LCD_String("Enter voltage");
```

```
                    LCD_Command(0xC0);
                    LCD_String(":");
                    _delay_ms(300);
                    LCD_Command(0x0F);
                    key_receive();
                    }
}
//printing values for user
//printing primary SWGA
LCD_String("PRI SWGA:");
P_SWGA=get_value();
char out_str[7]="0";
itoa(RAM_SWGA[P_SWGA],out_str,10);
LCD_Command(0xC0);
LCD_String(out_str);
key_enter_receive();


//printing secondary SWGA
for(int i=0;i<ACT;i++)
{
        LCD_Command(0X80);
        LCD_String("SEC SWGA:");
        LCD_Command(0xC0);
        S_SWGA=get_value();
        _delay_ms(1000);
        char out_str3[7]="0";
        itoa(S_SWGA,out_str3,10);
        LCD_String(out_str3);
        key_enter_receive();
        _delay_ms(500);
```

```
}
//printing core area
LCD_Command(0X80);
LCD_String("core area:");
C_AR=get_value();
out_str[7]="0";
itoa(C_AR,out_str,10);
LCD_Command(0xC0);
LCD_String(out_str);
key_enter_receive();

//printing ei lamination
LCD_Command(0X80);
LCD_String("EI LAMINATION:");
EI_LA=get_value();
LCD_Command(0xC0);
LCD_String(RAM_EILAM[EI_LA]);
key_enter_receive();

//Bobbing measuring unit
LCD_Command(0X80);
LCD_String("Place the Bobbin");
key_enter_receive();

LCD_Command(0X0C);
LCD_Command(0X80);
LCD_String("Rotate Bobbin");
key_enter_receive();

//z length measuring
LCD_Command(0X80);
```

```
LCD_String("adjust bobbin");
key_enter_receive();

Bob_result=get_value();
if(Bob_result==1)
{
        LCD_Command(0X80);
        LCD_String("Bobbin is suitable");
        LCD_Command(0XC0);
        LCD_String("le");
        key_enter_receive();
}
else
{
        LCD_Command(0X80);
        LCD_String("Bobbin isnot sui");
        LCD_Command(0XC0);
        LCD_String("table");
        key_enter_receive();
}

LCD_Command(0X80);
LCD_String("Bobbin measuring");
LCD_Command(0XC0);
LCD_String("completed");
key_enter_receive();

//starting the winding
LCD_Command(0X80);
LCD_String("Press # to start");
LCD_Command(0XC0);
```

```
        LCD_String("winding!");
        key_enter_receive();
        if(ACT>1)
        {
                win_result=get_value();
                if(win_result==10)
                {
                        LCD_Command(0X80);
                        LCD_String("Insert next coil");
                        key_enter_receive();
                }
                else if(win_result==30)
                {
                        LCD_Command(0X80);
                        LCD_String("Winding");
                        LCD_Command(0XC0);
                        LCD_String("completed!");
                        key_enter_receive();
                }
        }
        else
        {
                win_result=get_value();
                LCD_Command(0X80);
                LCD_String("Winding");
                LCD_Command(0XC0);
                LCD_String("completed!");
                key_enter_receive();
        }
}
```

slave_SPI.h

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <string.h>

#define MOSI 5                  /* Define SPI bus pins */
#define MISO 6
#define SCK 7
#define SS 4
#define SS_Enable PORTB &= ~(1<<SS)             /* Define Slave enable */
#define SS_Disable PORTB |= (1<<SS)             /* Define Slave disable */

void SPI_Init()                         /* SPI Initialize function */
{
        DDRB &= ~((1<<MOSI)|(1<<SCK)|(1<<SS));          /* Make MOSI, SCK, SS pin
direction as input pins */
        DDRB |= (1<<MISO);                      /* Make MISO pin as output pin */
        SPCR = (1<<SPE);                        /* Enable SPI in slave mode */
}

char SPI_Receive()                      /* SPI Receive data function */
{
        while(!(SPSR & (1<<SPIF)));             /* Wait till reception complete */
        return(SPDR);                   /* return received data */
}
```

```
int get_value()
{

        int value;

        SPI_Init();
        value=SPI_Receive();
        return value;

}
```

Name of student: SANJAYA B.H (204189U)

1. Responsible part - Winding unit

Learned to code atmega32 microcontroller using Atmel studio Software and learned to use Proteus software to simulate the circuit. I was aware of how to use the Stepper motor, and Servo motor. In the winding process, I have designed and used 2 robotic hands to guide the copper wires along the bobbin and 2 robotic hands to guide the tape along the bobbin in the winding process. Each hand has one stepper motor and servo motor. There are 4 stepper motors and 4 servo motors in robotic hands. Then I designed and used another separate unit for the tapping process. There are 2 stepper motors used in that unit for the tapping purpose. Also, I have the responsibility to synchronize all these hands 'movements with the bobbin rotation. Then we have used 4 demultiplexers to connect all these stepper motors and 1 demultiplexer to the all-servo motors. I was aware of the working mechanism and pins of all the components that I have used. I did the stepper motor and servo motor coding part and simulated the circuit using proteus software. I drew the schematic diagram and PCB design of those components.

2. Specification

Stepper motor

One revolution of the stepper motor is divided into a discrete number of steps, in our case we have chosen 200 steps per revolution, and the motor needs a separate pulse for each step. The stepper motors can take one step at a time with the same size. We have chosen the Nema-17 Stepper motor as our stepper motor. It is a 12V stepper motor with a 1.8^0 step angle which can operate at high torque. It takes 200 per revolution.

Servo motor

As the servo motor, we have chosen the M0090 servo motor which can operate at high tension. It can move from 0 to $+90^0$ or 0to -90^0. It can move 180^0 in our scenario. It has a 4V-6V supply voltage which can directly connect to the microcontroller. It weights 14g, and its dimensions are 22.4x12.5x22. 8mm. It has a speed of 80ms/60° (6V).

L293D motor driver IC

The L293D IC works as a driver circuit for the Stepper motors. It receives signals from the Atmega32 and transmits the relative signal to the Stepper motors. Also, it works as a protection for the motors. Because in case of a sudden high tension that can't be handled by the motor as these motors are worked in high tension, that driver circuit can protect the components. It is worked as a two bridge to the stepper motors. This IC has one supply voltage (V_S, one logic supply voltage (V_{SS}), 4 inputs and 4 outputs with 2 enable pins. So here one L293D IC has been used to connect one stepper motor with Atmega32.

CD74HC4051 IC

The CD74HC4051 can be used as an 8:1 Demultiplexer, that can give one input to separate 8-channels based on the channel select pins. In our scenario, the eight outputs channels are A_0, A_1, A_2, A_3, A_4, A_5, A_6, and A_7. The output on the single channel is decided based on the channel select pins of S_0, S_1, and S_2. According to the select pins relevant output channel will be selected and output will be passed. The Maximum recommended V_{cc} is 6V.

CD4052 IC

The CD4052 can be used as an 4:1 Demultiplexer, that can give one input to separate 4 channels based on the channel select pins. In our scenario, the four outputs channels are Y0,Y1,Y2. The output on the single channel is decided based on the channel select pins of A, B, and C.
According to the select pins relevant output channel will be selected
and output will be passed. The Maximum recommended Vcc is 5V.Vee is -5V and Vss is 0V.

3. Technique

1. Each robotic hand is moving along through the metal thread bar.
2. Each thread bar is attached to a separate stepper motor.
3. Each thread bar has a 0.5mm pitch.
4. Each stepper motor has a 1.8^0 step angle.
5. Each stepper motor is using full step sequence in rotating.
6. In one sequence it completes a 7.2^0 angle.
7. For one revolution of thread in the metal bar it needs 50 sequences in full step.
8. In one sequence hand is moving 0.01 mm along the thread bar.
9. 0.01mm is the smallest wire gauge that is used in the machine.
10. In every bobbin rotation hand is moving equally to the gauge of wire.
11. All the primary, secondary, tape arm 1 and tape arm 2 are using this technique to synchronize the moving distance with the bobbin rotation.
12. After the primary winding tape arms are used to cover the bobbin with insulating tape.
13. For that servo motors are used to bend the arms to stick the tapes and cut the tapes.
14. Then the secondary winding is started and at the tapping, points tapping control unit does the tapping process.
15. In the tapping control unit two stepper motors are used to move the tapping control unit and rotate the tapping wheel.
16. Tapping control unit is used to tie the tapping point of winding to the tapping pins to get the user desired voltages.
17. After the tapping process again tape arms are used to cover the crossed tapping wire along the bobbin.

18. After the secondary winding bobbin is covered with the insulating tape again and all the arms are moved to the previous starting locations.

4. Schematic Diagram

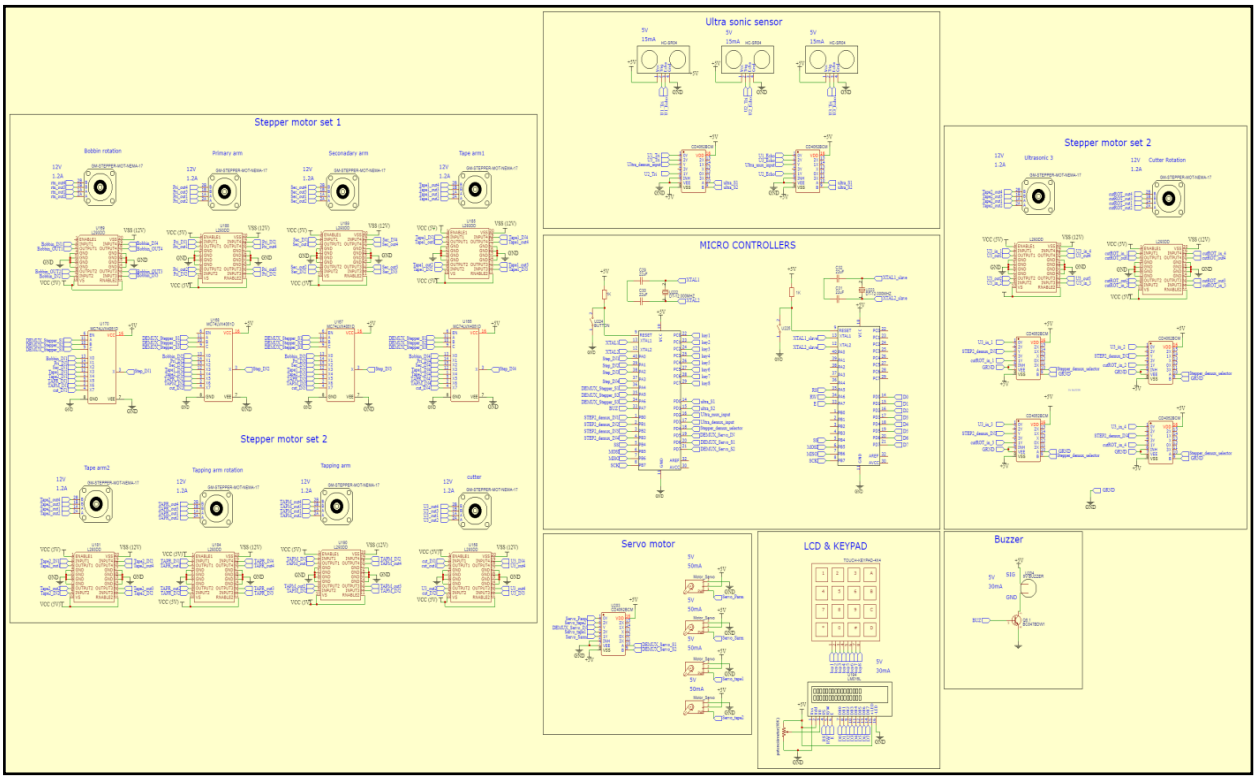

Figure 25

5. PCB Design

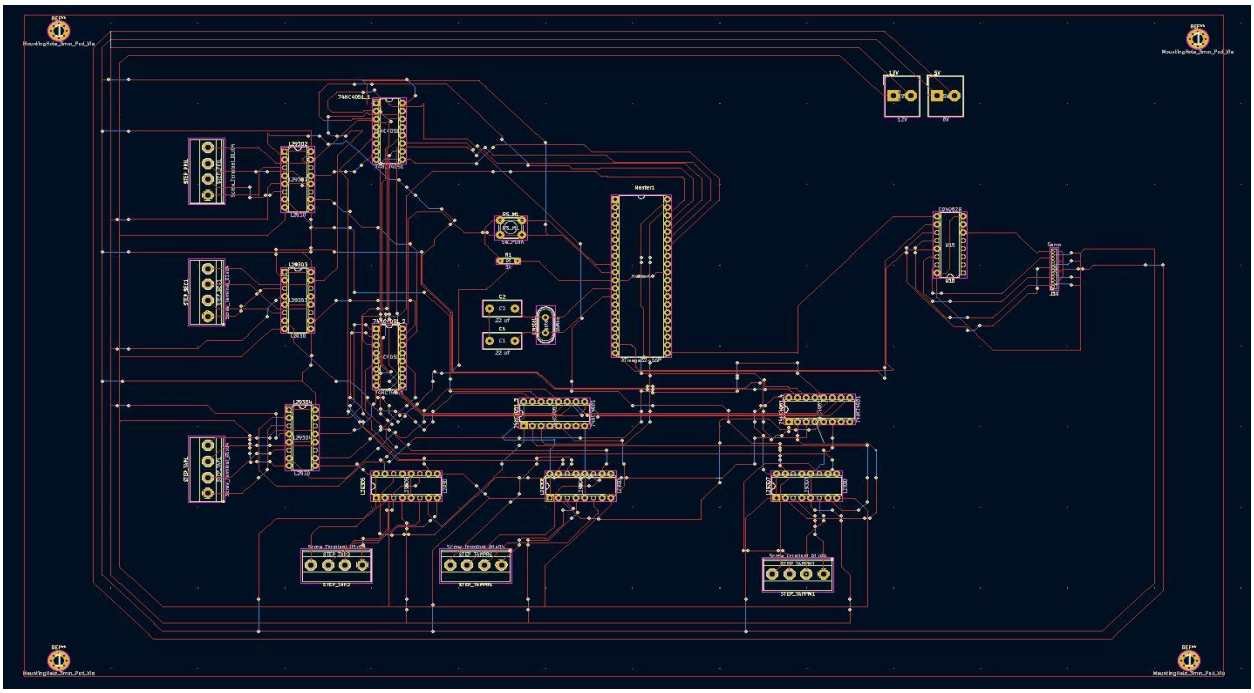

Figure 26

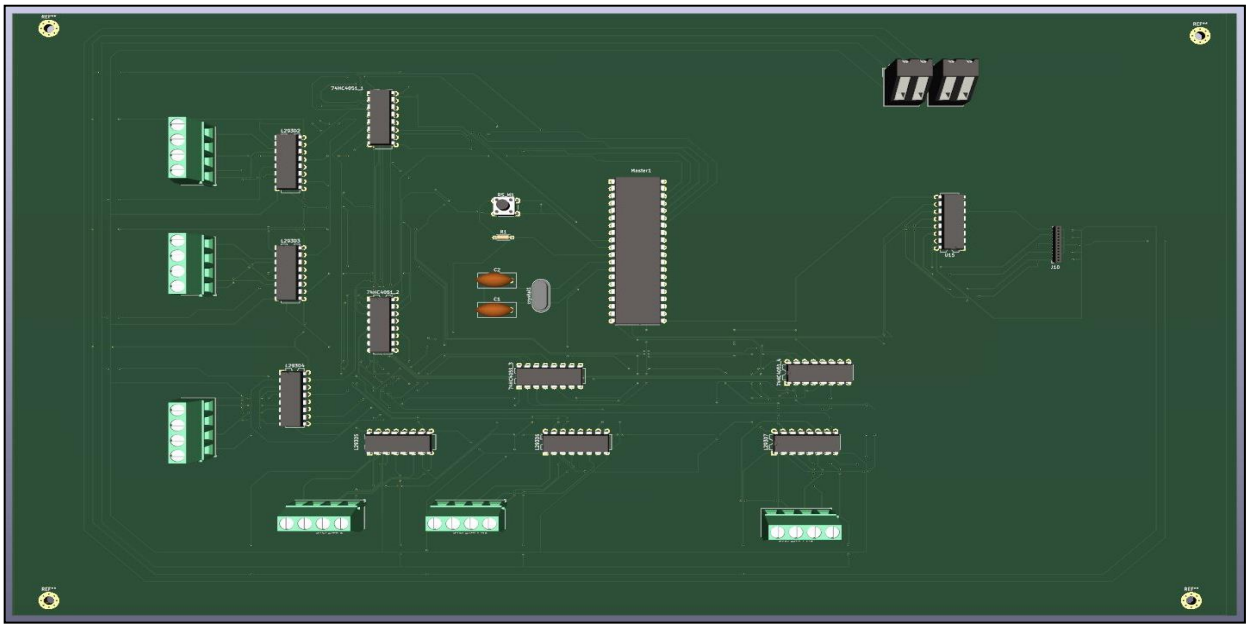

Figure 27

6. Proteus Simulation

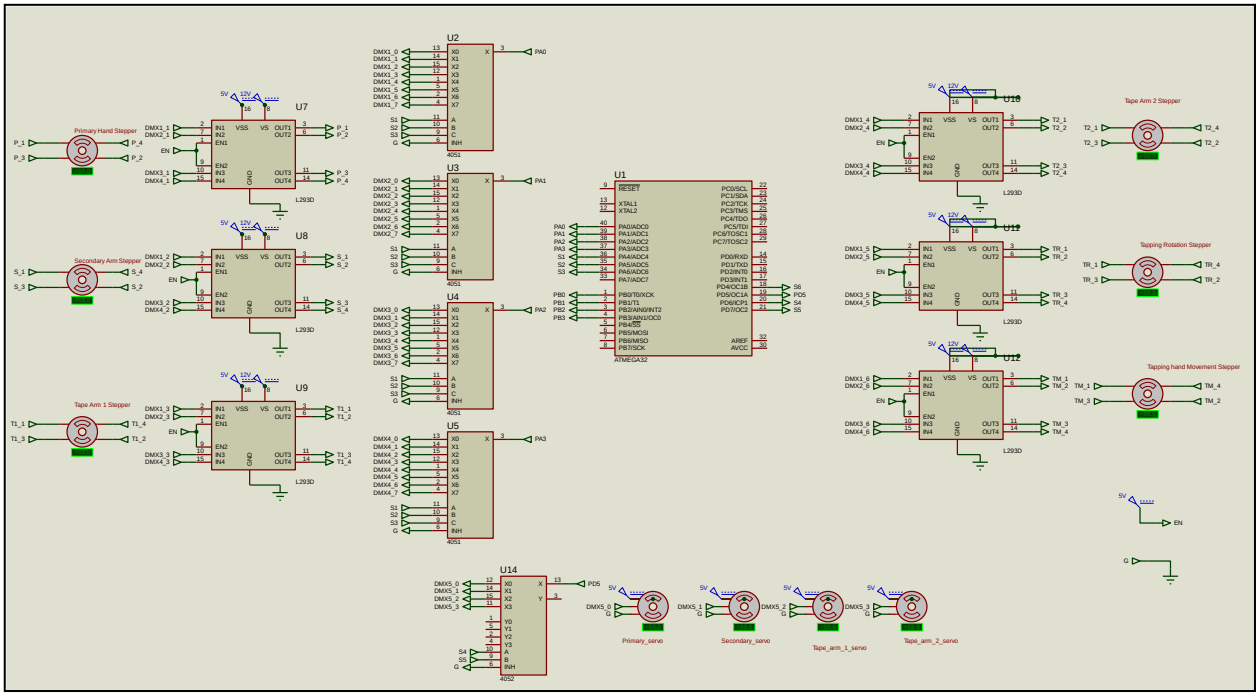

Figure 28

7. Code

```
#ifndef WINDING_PROCESS_H_
#define WINDING_PROCESS_H_
#define F_CPU 8000000UL             /* Define CPU Frequency 8MHz */
#include <avr/io.h>                 /* Include AVR std. library file */
#include <util/delay.h>                     /* Include delay header file */
#include "motors.h"

void primary_hand(int g)
{
        SM1_PORT = (((SM1_PORT&~(1<<s3))&~(1<<s2))|(1<<s1));
        stepper_1(g);   //          g = gauge of wire / 0.01mm

}
void secondary_hand(int g)
{
        SM1_PORT = (((SM1_PORT&~(1<<s3))|(1<<s2))&~(1<<s1));     //selecting secondary
hand stepper
        stepper_1(g);   //          g = gauge of wire / 0.01mm

}
void secondary_hand_reverse(int g)
{
        SM1_PORT = (((SM1_PORT&~(1<<s3))|(1<<s2))&~(1<<s1));     //selecting secondary
hand stepper
        stepper_1_reverse(g); //          g = gauge of wire / 0.01mm

}

void primary_hand_reverse(int g)
```

```
{
        SM1_PORT = (((SM1_PORT&~(1<<s3))&~(1<<s2))|(1<<s1));     ////selecting primary
hand stepper
        stepper_1_reverse(g); //         g = gauge of wire / 0.01mm

}

void tape_fun_1(int count,int g)        //g=  how many times as 0.01
{
        int w=0.05;     //          w =width of the tape
        if(count*0.01<w)
        {
                bobbin();
                primary_hand(g);
        }
        else if ((count*0.01<w+0.1)&&(count*0.01<w+0.1))
        {
                bobbin();
                primary_hand(g);
                tape_arm_1_placing(2);          //          width of the tape+moving distance to the
bobbin starting point)/0.01mm

        }
        else
        {
                bobbin();
                primary_hand(g);
                tape_arm_1(g);
        }
}
```

```
void tape_arm_1_placing(int r)
{
        SM1_PORT = (((SM1_PORT&~(1<<s3))|(1<<s2))|(1<<s1));                    //
        selecting tape arm_1
        stepper_1(r);   //      tape arm 1 is moving to the starting point
        SERS_PORT = ((SERS_PORT|(1<<s5))&~(1<<s4));                    //      selecting tape
arm_1 servo
        servo_half();   //      bend down to stick the tape
}

void tape_arm_1(int g)
{
        SM1_PORT = (((SM1_PORT&~(1<<s3))|(1<<s2))|(1<<s1));
        stepper_1(g);
}

void tape_arm_1_reverse(int g)
{
        SM1_PORT = (((SM1_PORT&~(1<<s3))|(1<<s2))|(1<<s1));
        stepper_1_reverse(g);
}

void tape_fun_2(int count,int g)
{
        int w=0.05;     //      w =width of the tape
        if(count*0.01<w)
        {
                bobbin();
                primary_hand_reverse(g);
        }
        else if ((count*0.01<w+0.1)&&(count*0.01<w+0.1))
```

```
	{
		bobbin();
		primary_hand_reverse(g);
		tape_arm_2_placing(2);	//	(width of the tape+moving distance to the
bobbin starting point)/0.01mm
	}
	else
	{
		bobbin();
		primary_hand_reverse(g);
		tape_arm_2(g);
	}
}

void tape_arm_2_placing(int r)
{
	SM1_PORT = (((SM1_PORT|(1<<s3))&~(1<<s2))&~(1<<s1));		//
	selecting tape arm_2
	stepper_1(r);	//	tape arm 2 is moving to the starting point
	SERS_PORT = ((SERS_PORT|(1<<s5))|(1<<s4));		//	selecting tape arm_2
servo
	servo_half();	//	bend down to stick the tape
}

void tape_arm_2(int g)
{
	SM1_PORT = (((SM1_PORT|(1<<s3))&~(1<<s2))&~(1<<s1));
	stepper_1(g);
}

void tape_arm_2_reverse(int g)
```

```
{
	SM1_PORT = (((SM1_PORT|(1<<s3))&~(1<<s2))&~(1<<s1));
	stepper_1_reverse(g);
}


void tapping_process(int d,int shift,int side,int g)
{
	int l=0;
	if (shift%2==1)
	{
		l=side-d;	//d = distance move to the starting point of secondary hand
	}
	SM1_PORT = (((SM1_PORT&~(1<<s3))|(1<<s2))&~(1<<s1));	//	selecting
secondary hand stepper
	stepper_1_reverse(l+1);
	// distance 1=50000
	SERS_PORT = ((SERS_PORT&~(1<<s5))|(1<<s4));		//	selecting
secondary hand servo
	servo_half_stop();
	SM1_PORT = (((SM1_PORT|(1<<s3))&~(1<<s2))|(1<<s1));		//
	selecting tapping rotation stepper
	stepper_1_first_half();
	SM1_PORT = (((SM1_PORT|(1<<s3))|(1<<s2))&~(1<<s1));		//
	selecting tapping movement stepper
	stepper_1(2);
	// move pre-defined distance for the location(2000)
	SM1_PORT = (((SM1_PORT&~(1<<s3))&~(1<<s2))|(1<<s1));	//	selecting
secondary hand stepper
	stepper_1(2);
	// move pre-defined distance for the location(2000)
```

```
	SM1_PORT = (((SM1_PORT|(1<<s3))&~(1<<s2))|(1<<s1));		//
	selecting tapping rotation stepper
	stepper_1_second_half();
	SERS_PORT = ((SERS_PORT&~(1<<s5))|(1<<s4));		//
	selecting secondary hand servo
	servo_half_stop_reverse();
	SM1_PORT = (((SM1_PORT&~(1<<s3))&~(1<<s2))|(1<<s1));	//	selecting
secondary hand stepper
	stepper_1_reverse(l+5);
	//	move to the previous location (48000)
	SM1_PORT = (((SM1_PORT|(1<<s3))&~(1<<s2))|(1<<s1));		//
	selecting tapping rotation stepper
	stepper_1_reverse(1);		//
rotate to the previous location
	SM1_PORT = (((SM1_PORT|(1<<s3))|(1<<s2))&~(1<<s1));		//
	selecting tapping movement stepper
	stepper_1_reverse(2);		//
rotate to the previous location
	if (shift%2==0)
	{
		tape_arm_2_placing(d);		//placing tape arm 2
	}
	else
	{
		tape_arm_1_placing(d);		//placing tape arm 1
	}
}

void covering_tape(int selection,int g)
{
	if (selection ==1)	// check to use tape arm_1
```

```
	{
		tape_arm_1_reverse(g);
	}
	else
	{
		tape_arm_2(g);		// check to use tape arm_2
	}
}


void ending_tape_1(int d,int g)
{
	for (int i=0;i<d;i++)
	{
		tape_arm_1_reverse(g);		//move to the previous starting location of tape arm
2
	}
}

void ending_tape_2(int d,int g)
{
	for (int i=0;i<d;i++)
	{
		tape_arm_2_reverse(g);		//move to the previous starting location of tape arm
2
	}
}

void tape_1_cutting()
{
```

```
	SERS_PORT = ((SERS_PORT|(1<<s5))&~(1<<s4));		//	selecting tape
arm_1 servo
	servo_half();	//	bend down to cut the tape
}

void tape_2_cutting()
{
	SERS_PORT = ((SERS_PORT|(1<<s5))|(1<<s4));		//	selecting tape arm_2
servo
	servo_half();	//	bend down to cut the tape
}
void ending_tape(int side,int g)
{
	tape_arm_2_placing(0);		//placing tape arm 2 and stick the tape
	for (int i=0;i<side;i++)
	{
		tape_arm_2(g);			//stick the tape all over the side
	}
	for (int j=0;j<side;j++)
	{
		tape_arm_2_reverse(g);	// stick again all over the side
	}
}
void primary_winding_end(int d,int g)
{
	int x=1;			//	distance for the cutter from bobbin
	tape_arm_1_reverse(x);
	SM1_PORT = (((SM1_PORT&~(1<<s3))&~(1<<s2))|(1<<s1));	//selecting primary
arm stepper
	stepper_1_reverse(d*g+x);
```

```
	SERS_PORT = ((SERS_PORT&~(1<<s5))&~(1<<s4));		//	selecting
primary arm servo
	servo_half_stop();
	cutting_process();
	SERS_PORT = ((SERS_PORT&~(1<<s5))&~(1<<s4));		//	selecting
primary arm servo
	servo_half_stop_reverse();
	SM1_PORT = (((SM1_PORT&~(1<<s3))&~(1<<s2))|(1<<s1));
	stepper_1(3);
moving to the defined end point
	tape_arm_1(3);
	// moving tape arm to the defined end point
}

void secondary_winding_end(int d,int g)
{
	int x=1;
	//	distance for the cutter from bobbin
	tape_arm_1_reverse(x);
	SM1_PORT = (((SM1_PORT&~(1<<s3))|(1<<s2))&~(1<<s1));	//	selecting
secondary arm
	stepper_1_reverse(d*g+x);
	SERS_PORT = ((SERS_PORT&~(1<<s5))|(1<<s4));
	selecting secondary arm servo
	servo_half_stop();
	cutting_process();
	SERS_PORT = ((SERS_PORT&~(1<<s5))|(1<<s4));
	selecting secondary arm servo
	servo_half_stop_reverse();
	SM1_PORT = (((SM1_PORT&~(1<<s3))|(1<<s2))&~(1<<s1));
```

```
	stepper_1(3);                                                    //
moving secondary arm to the defined end point
	tape_arm_1(3);
	// moving tape arm to the defined end point
}

void secondary_winding_no_tap(int gauge,int round,int side)
{
	int shift=0;
	int end_round=0;
	while (1)
	{
		if(round>side)
		{
			if((round-(shift*side))>side)
			{
				if((shift%2)==0)
				{
					for(int i=0;i<side;i++)
					{
						bobbin();
						secondary_hand(gauge);
					}
					shift++;
				}
				else
				{
					for(int i=0;i<side;i++)
					{
						bobbin();
						secondary_hand_reverse(gauge);
```

```
                }
                shift++;
            }
        }
        else
        {
            if((shift%2)==0)
            {
                for(int i=0;i<(round-(shift*side));i++)
                {
                    tape_fun_1(i+1,gauge);
                    end_round=i;
                }
                tape_1_cutting();
                ending_tape_1(end_round,gauge);
                break;
            }
            else
            {
                for(int i=0;i<(round-(shift*side));i++)
                {
                    tape_fun_2(i+1,gauge);
                    end_round=i;
                }
                tape_2_cutting();
                ending_tape_2(end_round,gauge);
                break;
            }
        }
    }
    else
```

```
		{
			for(int i=0;i<round;i++)
			{
				bobbin();
				secondary_hand(gauge);
				tape_fun_1(i+1,gauge);
				end_round=i;
			}
			tape_1_cutting();
			ending_tape_1(end_round,gauge);
			break;
		}
	}
	secondary_winding_end(end_round,gauge);
	ending_tape(side,gauge);
}

void primary_winding(int gauge,int round,int side)
{
	int shift=0;
	int end_round=0;
	while (1)
	{
		if(round>side)
		{
			if((round-(shift*side))>side)
			{
				if((shift%2)==0)
				{
					for(int i=0;i<side;i++)
					{
```

```
                bobbin();
                primary_hand(gauge);
            }
            shift++;
        }
        else
        {
            for(int i=0;i<side;i++)
            {
                bobbin();
                primary_hand_reverse(gauge);
            }
            shift++;
        }
}
else
{
        if((shift%2)==0)
        {
            for(int i=0;i<(round-(shift*side));i++)
            {
                tape_fun_1(i+1,gauge);
                end_round=i;
            }
            tape_1_cutting();
            ending_tape_1(end_round,gauge);
            break;
        }
        else
        {
            for(int i=0;i<(round-(shift*side));i++)
```

```
                                        {
                                                tape_fun_2(i+1,gauge);
                                                end_round=i;
                                        }
                                        tape_2_cutting();
                                        ending_tape_2(end_round,gauge);
                                        break;
                                }
                        }
                }
                else
                {
                        for(int i=0;i<round;i++)
                        {
                                bobbin();
                                primary_hand(gauge);
                                tape_fun_1(i+1,gauge);
                        }
                        tape_1_cutting();
                        ending_tape_1(end_round,gauge);
                        break;
                }
        }
        primary_winding_end(end_round,gauge);
        ending_tape(side,gauge);
}

void secondary_winding(int gauge,int side,int* passed_tap_arr,int no_of_taps)
{
        int round=0;
        int shift = 0;
```

```
int end_round=0;
int k = 0;
for (int j = 0; j < no_of_taps; j++)
{
        round = passed_tap_arr[j];
        while (1)
        {
                if (round > side)
                {
                        if ((round - (shift * side)) > side)
                        {
                                if ((shift % 2) == 0)
                                {
                                        for (int i = 0; i < side; i++)
                                        {
                                                if (k > 0)
                                                {
                                                        bobbin();
                                                        secondary_hand(gauge);
                                                        covering_tape(2, gauge);
                                                        k--;
                                                } else
                                                {
                                                        bobbin();
                                                        secondary_hand(gauge);
                                                        //k--;
                                                }

                                        }
                                        shift++;
                                } else
```

```
        {
            for (int i = 0; i < side; i++)
            {
                if (k > 0)
                {
                    bobbin();
                    secondary_hand_reverse(gauge);
                    covering_tape(1, gauge);
                    k--;
                } else
                {
                    bobbin();
                    secondary_hand_reverse(gauge);
                    //k--;
                }
            }
            shift++;
        }
    }
    else
    {
        if ((shift % 2) == 0)
        {
            for (int i = 0; i < (round - (shift * side)); i++)
            {
                tape_fun_1(i + 1, gauge);
                k = i;
                end_round=i;
            }
            tape_1_cutting();
            ending_tape_1(end_round,gauge);
```

```
                                        tapping_process(k, shift, side,gauge);    // k=
remaining rounds to cover the crossed tapping wire
                                        break;
                                } else
                                {
                                        for (int i = 0; i < (round - (shift * side)); i++)
                                        {
                                                tape_fun_2(i + 1, gauge);
                                                k = i;
                                                end_round=i;
                                        }
                                        tape_2_cutting();
                                        ending_tape_2(end_round,gauge);
                                        tapping_process(k, shift, side,gauge);    // k=
remaining rounds to cover the crossed tapping wire
                                        break;
                                }
                        }

                }
                else
                {
                        for (int i = 0; i < round; i++)
                        {
                                bobbin();
                                secondary_hand(gauge);
                                tape_fun_1(i + 1, gauge);
                                k = i;
                                end_round=i;
                        }
                        tape_1_cutting();
```

```
                                ending_tape_1(end_round,gauge);
                                tapping_process(k, shift, side, gauge);    // k= remaining rounds to
cover the crossed tapping wire
                                break;
                        }
                }
        }
        secondary_winding_end(end_round,gauge);
        ending_tape(side,gauge);
}

#endif /* WINDING_PROCESS_H_ */
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#define m1 0
#define m2 1
#define m3 2
#define m4 3
#define m5 0
#define m6 1
#define m7 2
#define m8 3
#define s1 4
#define s2 5
#define s3 6
#define s4 6
#define s5 7
#define s6 4
#define SM1_PORT PORTA
#define SM2_PORT PORTD
```

```
#define SERS_PORT PORTD
#define M1_PORT PORTA
#define M2_PORT PORTB

void stepper_1(int n)
{
	int period;
	period = 1000;
	DDRA = 0xFF;
	while (1)
	{
		M1_PORT = ((((M1_PORT&~(1<<m4))&~(1<<m3))&~(1<<m2))&~(1<<m1));
		for(int i=0;i<n;i++)
		{
			M1_PORT =
((((M1_PORT|(1<<m4))&~(1<<m3))&~(1<<m2))|(1<<m1));
			_delay_ms(period);
			M1_PORT =
((((M1_PORT|(1<<m4))|(1<<m3))&~(1<<m2))&~(1<<m1));
			_delay_ms(period);
			M1_PORT =
((((M1_PORT&~(1<<m4))|(1<<m3))|(1<<m2))&~(1<<m1));
			_delay_ms(period);
			M1_PORT =
((((M1_PORT&~(1<<m4))&~(1<<m3))|(1<<m2))|(1<<m1));
			_delay_ms(period);
		}
		break;
	}
}
```

```
void stepper_1_first_half()
{
	int period;
	period = 1000;
	DDRA = 0xFF;
	M1_PORT = ((((M1_PORT&~(1<<m4))&~(1<<m3))&~(1<<m2))&~(1<<m1));
	_delay_ms(100);
	M1_PORT = ((((M1_PORT|(1<<m4))&~(1<<m3))&~(1<<m2))|(1<<m1));
	_delay_ms(period);
	M1_PORT = ((((M1_PORT|(1<<m4))|(1<<m3))&~(1<<m2))&~(1<<m1));
	_delay_ms(period);
}

void stepper_1_first_half_reverse()
{
	int period;
	period = 1000;
	DDRA = 0xFF;
	M1_PORT = ((((M1_PORT&~(1<<m4))&~(1<<m3))&~(1<<m2))&~(1<<m1));
	M1_PORT = ((((M1_PORT|(1<<m4))&~(1<<m3))&~(1<<m2))|(1<<m1));
	_delay_ms(period);
	M1_PORT = ((((M1_PORT&~(1<<m4))&~(1<<m3))|(1<<m2))|(1<<m1));
	_delay_ms(period);
}

void stepper_1_second_half()
{
	int period;
	period = 1000;
	DDRA = 0xFF;
	M1_PORT = ((((M1_PORT&~(1<<m4))|(1<<m3))|(1<<m2))&~(1<<m1));
```

```
        _delay_ms(period);
        M1_PORT = ((((M1_PORT&~(1<<m4))&~(1<<m3))|(1<<m2))|(1<<m1));
        _delay_ms(period);
}

void stepper_1_second_half_reverse()
{
        int period;
        period = 1000;
        DDRA = 0xFF;
        M1_PORT = ((((M1_PORT|(1<<m4))|(1<<m3))&~(1<<m2))&~(1<<m1));
        _delay_ms(period);
        M1_PORT = ((((M1_PORT&~(1<<m4))|(1<<m3))|(1<<m2))&~(1<<m1));
        _delay_ms(period);
}


void stepper_1_reverse(int n)
{
        int period;
        period = 1000;
        DDRA = 0xFF;
        while (1)
        {
                M1_PORT = ((((M1_PORT&~(1<<m4))&~(1<<m3))&~(1<<m2))&~(1<<m1));
                for(int i=0;i<n;i++)
                {
                        M1_PORT =
((((M1_PORT&~(1<<m4))|(1<<m3))|(1<<m2))&~(1<<m1));
                        _delay_ms(period);
```

```
                        M1_PORT =
((((M1_PORT|(1<<m4))|(1<<m3))&~(1<<m2))&~(1<<m1));
                        _delay_ms(period);
                        M1_PORT =
((((M1_PORT|(1<<m4))&~(1<<m3))&~(1<<m2))|(1<<m1));
                        _delay_ms(period);
                        M1_PORT =
((((M1_PORT&~(1<<m4))&~(1<<m3))|(1<<m2))|(1<<m1));
                        _delay_ms(period);
                }
                break;
        }
}

void servo_half()
{
        DDRD |= (1<<PD5);  /* Make OC1A pin as output */
        TCNT1 = 0;              /* Set timer1 count zero */
        ICR1 = 2499;            /* Set TOP count for timer1 in ICR1 register */

        /* Set Fast PWM, TOP in ICR1, Clear OC1A on compare match, clk/64 */
        TCCR1A = (1<<WGM11)|(1<<COM1A1);
        TCCR1B = (1<<WGM12)|(1<<WGM13)|(1<<CS10)|(1<<CS11);
        while(1)
        {
                OCR1A = 300;        /* Set servo shaft at 90° position */
                _delay_ms(1500);
                OCR1A = 187;        /* Set servo at +0° position */
                _delay_ms(1500);
                break;
        }
```

```
}

void servo_half_stop()
{
        DDRD |= (1<<PD5);  /* Make OC1A pin as output */
        TCNT1 = 0;         /* Set timer1 count zero */
        ICR1 = 2499;       /* Set TOP count for timer1 in ICR1 register */

        /* Set Fast PWM, TOP in ICR1, Clear OC1A on compare match, clk/64 */
        TCCR1A = (1<<WGM11)|(1<<COM1A1);
        TCCR1B = (1<<WGM12)|(1<<WGM13)|(1<<CS10)|(1<<CS11);
        while(1)
        {
                OCR1A = 300;       /* Set servo shaft at 90° position */
                _delay_ms(1500);
                break;
        }
}

void servo_half_stop_reverse()
{
        DDRD |= (1<<PD5);  /* Make OC1A pin as output */
        TCNT1 = 0;         /* Set timer1 count zero */
        ICR1 = 2499;       /* Set TOP count for timer1 in ICR1 register */

        /* Set Fast PWM, TOP in ICR1, Clear OC1A on compare match, clk/64 */
        TCCR1A = (1<<WGM11)|(1<<COM1A1);
        TCCR1B = (1<<WGM12)|(1<<WGM13)|(1<<CS10)|(1<<CS11);
        while(1)
        {
                OCR1A = 187;       /* Set servo at +0° position */
```

```
                _delay_ms(1500);
                break;
        }
}

void stepper_2()
{
        int period;
        period = 1000;
        DDRB = 0xFF;
        while (1)
        {
                M2_PORT = ((((M2_PORT&~(1<<m8))&~(1<<m7))&~(1<<m6))&~(1<<m5));
                M2_PORT = ((((M2_PORT|(1<<m8))&~(1<<m7))&~(1<<m6))|(1<<m5));
                _delay_ms(period);
                M2_PORT = ((((M2_PORT&~(1<<m8))&~(1<<m7))|(1<<m6))|(1<<m5));
                _delay_ms(period);
                break;
        }
}
```

Name of student: SENEVIRATHNE Y.M.N.N (204197R)

1. Responsible part - Bobbin measuring unit

Learned to code atmega32 microcontroller using Atmel studio Software and learned to use Proteus software to simulate the circuit. I was aware of how to get the measurement of a selected bobbin. For that purpose, I used the 3 Ultrasonic sensors and a stepper motor. For the winding purpose, we want to know whether the selected bobbin is suited or not. Two ultrasonic sensors are used to measure the width of the bobbin and measure edge height of the bobbin. And Another sensor was used to measure the length of the bobbin with aid of a stepper motor. I was aware of the working mechanism and pins of all the components that I have used. I did the Ultrasonic sensors and stepper motor coding part and simulated the circuit using proteus software. I drew the schematic diagram and PCB design of those components.

2. Specification

Ultrasonic sensors

An ultrasonic sensor emits a sound pulse in the ultrasonic range. This sound pulse propagates at the speed of sound through air (about 344 meters per second) until the sound pulse encounters an object. The sound pulse bounces off the object and is returned in reverse to the sensor where this "echo" is received. By measuring the time, it takes for the sound pulse to travel from the sensor to the object and back to the sensor. The distance to the object can be calculated very accurately. So, I used 3 ultrasonic sensors for the distance measurements.

Stepper motor

Every revolution of the stepper motor is divided into a discrete number of steps, in many cases 200 steps, and the motor must be sent a separate pulse for each step. The stepper motor can only take one step at a time and each step is the same size. Since each pulse causes the motor to rotate at a precise angle, typically 1.8°, the motor's position can be controlled without any feedback mechanism. As the digital pulses increase in frequency, the step movement changes into the continuous rotation, with the speed of rotation directly proportional to the frequency of the

pulses. Here we need to place one ultrasonic sensor according to the bobbin size. For that purpose, I used a stepper motor to move an ultrasonic sensor with aid of thread bars.

L293D motor driver IC

The L293D IC receives signals from the Atmega32 and transmits the relative signal to the motors. It has two voltage pins, one of which is used to draw current for the working of the L293D and the other is used to apply voltage to the motors. The L293D switches its output signal according to the input received from the Atmega32. For Example: If the Atmega32 sends a 1(digital high) to the Input Pin of L293D, then the L293D transmits a 1(digital high) to the motor from its Output Pin. An important thing to note is that the L293D simply transmits the signal it receives. It does not change the signal in any case. So here one L293D IC has been used to connect a stepper motor with Atmega32.

CD4052 IC

The CD4052 can be used as a 4:1 Multiplexer, that is it can take inputs from 4-channel and convert it to single channel output based on the channel select pins. In our case, the four Input channels are X_0Y_0, X_1Y_1, X_2Y_2 and X_3Y_3 and the single output channel is X, Y. The output on the single channel is decided based on the channel select pins A and B. Also, CD4052 can be used as a 1:4 Demultiplexer, that is it can take one input and provide it to either one of the 4 output channels based on the channel select pins. Here the input pins will be X and Y. The output pins can either be X_0, Y_0 or X_1, Y_1 or X_2, Y_2 or X_3, Y_3 based on the value set on A and B pins. Two CD4052 IC are used to connect ultrasonic sensors to the atmega32. Select pins A and B are connected to D_0 and D_1 of the Atmega32 respectively. And X and Y pins are connected to the D_2 and D_3 of the Atmega32 respectively**.** Also, 4 CD4052 IC are used to connect pins B_0, B_1, B_2, and B_3 of Atmega32 to an L293D motor driver IC.

3. Technique

1. User must place the bobbin on the left corner
2. u1 ultrasonic sensor moves to the left with aid of a stepper motor and thread bars to measure the distance(Z_1) to the wall.
3. Stepper motor works until the above-measured distance is less than a fixed distance (this fixed value(Z_1) is included in the code). If the above-measured distance is less than that fixed value, then the motor stopped and gets the Z_2 distance.
4. Then the u3 ultrasonic sensor measures the edge height of the bobbin (X_3 distance) and the u2 ultrasonic sensor measures the X_2 distance respectively.
5. User rotates the bobbin.
6. Then the u3 ultrasonic sensor measures the Y_3 distance and the u2 ultrasonic sensor measures the Y_2 distance respectively.
7. Using the above measurements displayed whether that selected bobbin is suited for the winding process.

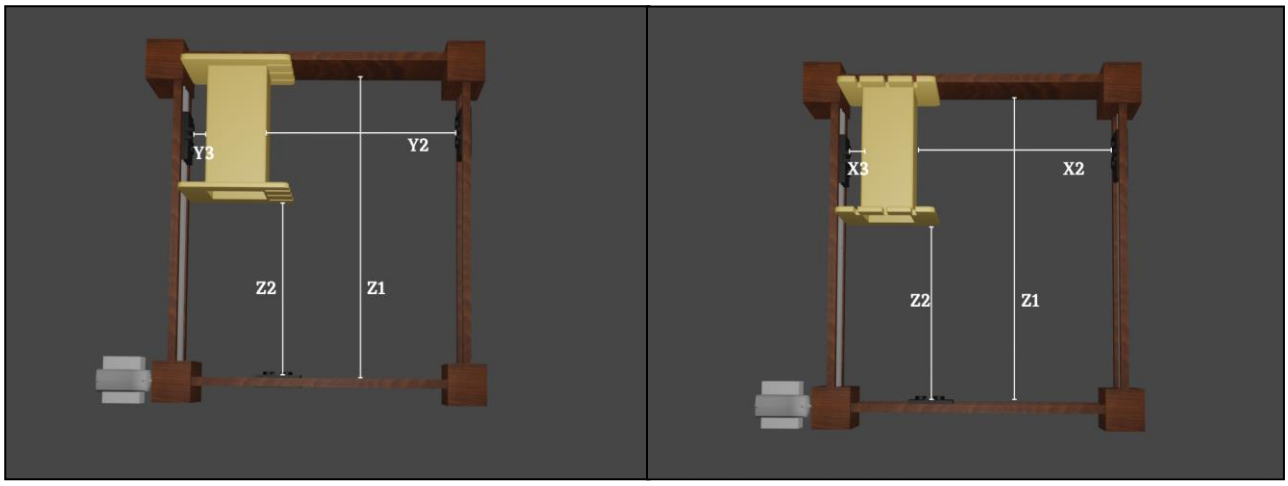

Figure 29

4. Schematic Diagram

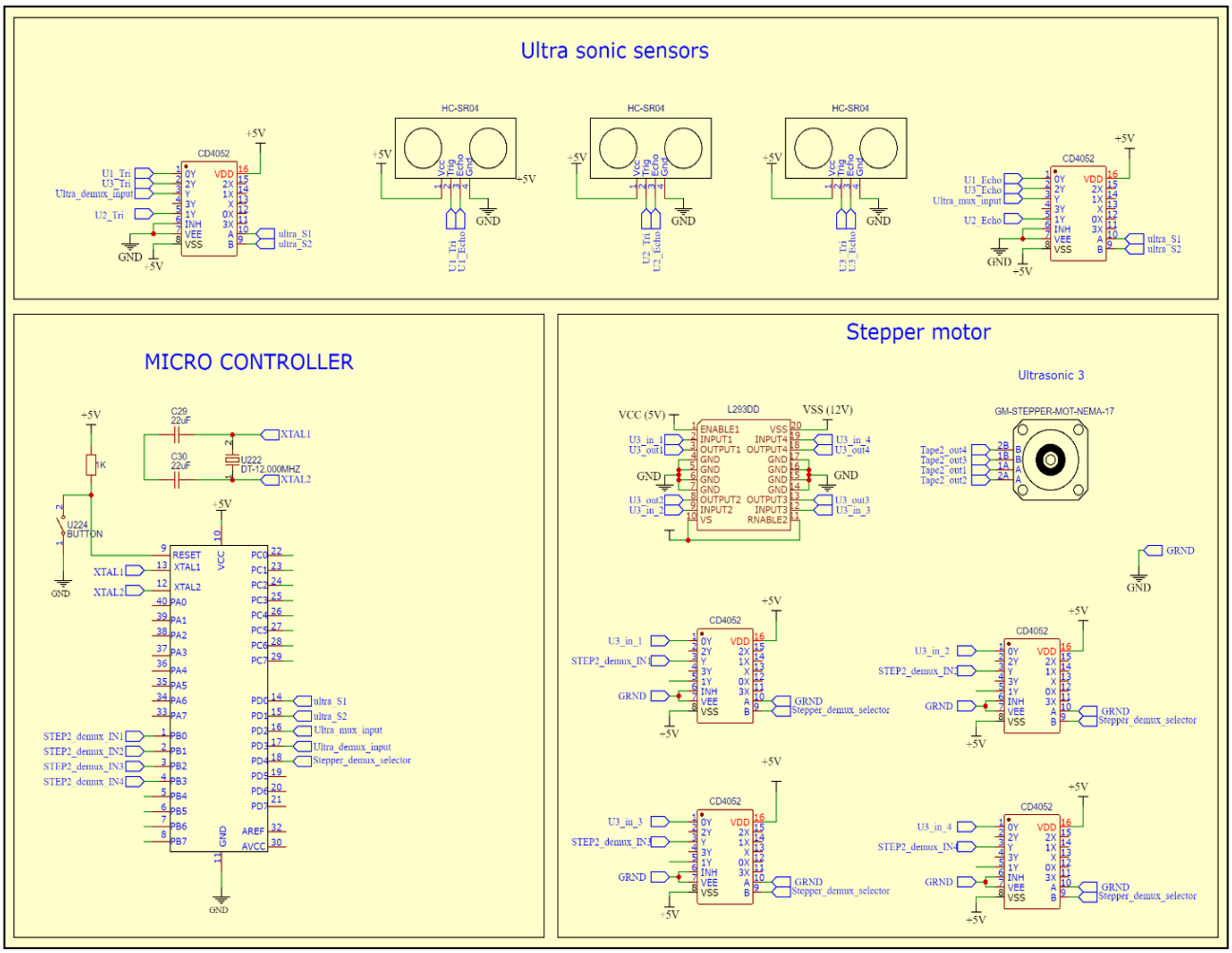

Figure 30

5. PCB Design

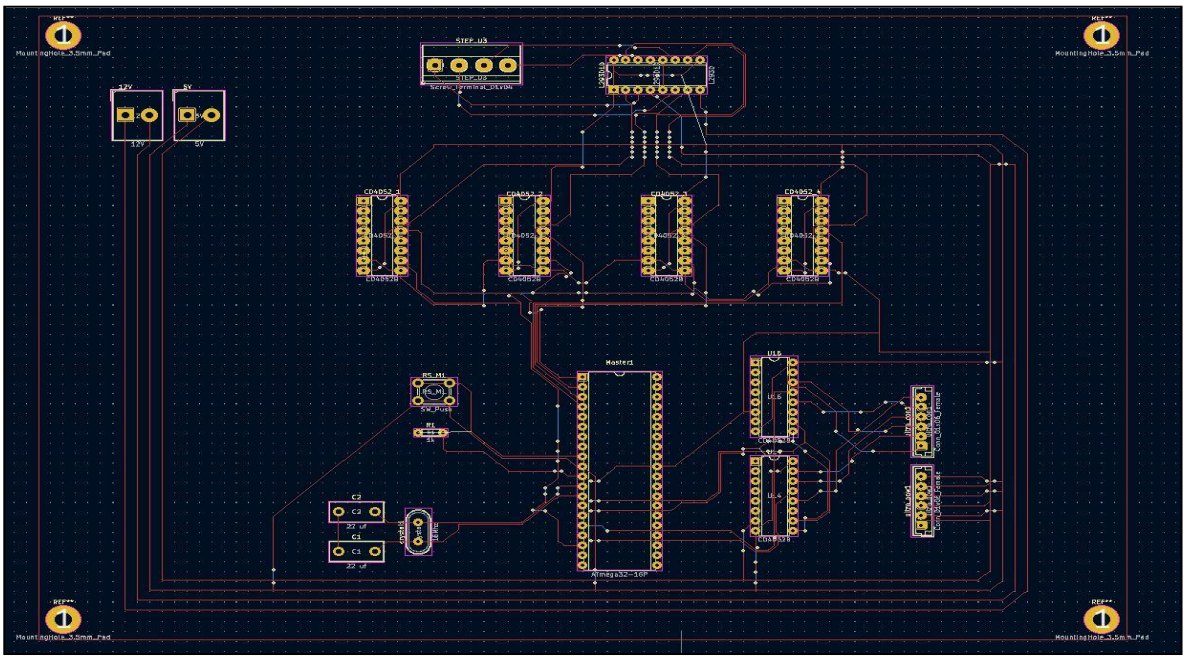

Figure 31

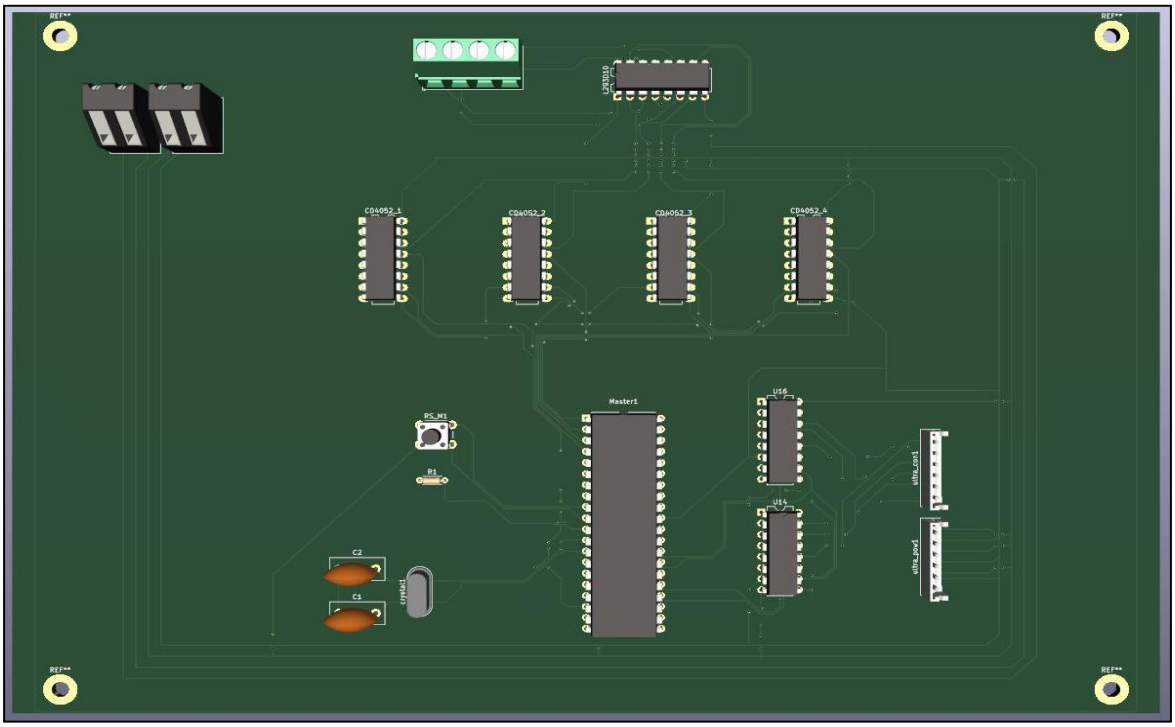

Figure 32

6. Proteus Simulation

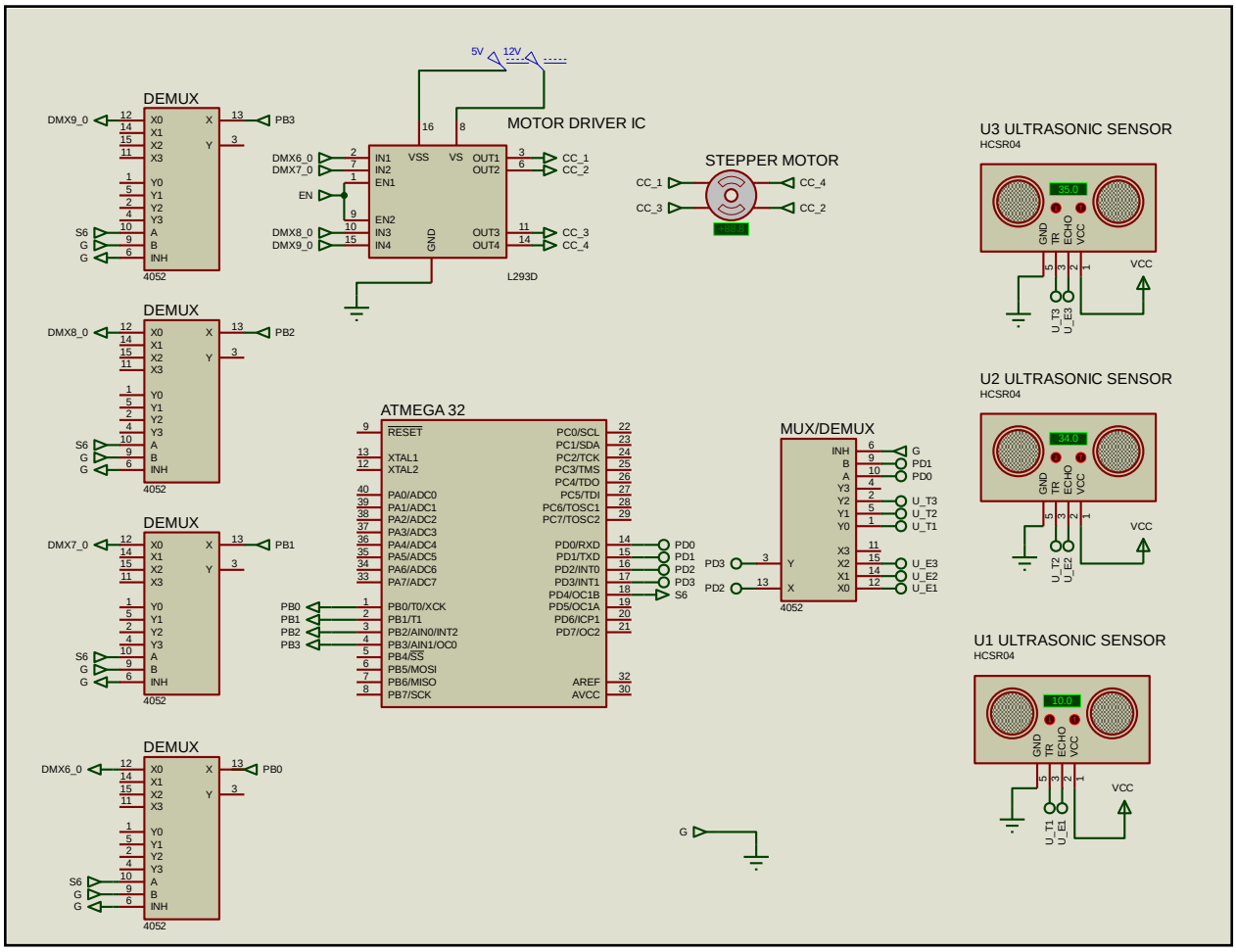

Figure 33

7. Code

main.c

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include "ultrasonic.h"

int main(void)
{

        int x1_distance,
        x2_distance,
        y1_distance,
        y2_distance,
        z1_distance,
        cal_core,               //calculated core area using ultra-sonic sensors
        x_measurement,                  //core area width
        y_measurement,                  //core area length
        z_measurement,                  //Bobbin length
        x_fix=30,
        y_fix=30,
        z_fix=30;
```

```
        //Place the bobbin

        //Measuring x length
        x1_distance=ultrasonic_1();
        x2_distance=ultrasonic_2();

        x_measurement=x_fix-x1_distance-x2_distance;

        //rotate the bobbin

        //Measuring y length
        y1_distance=ultrasonic_1();
        y2_distance=ultrasonic_2();

        y_measurement=y_fix-y1_distance-y2_distance;

        //start measuring z length

        //Measuring Z length
        z1_distance=ultrasonic_3();

        z_measurement=z_fix-z1_distance;

        //Bobbin Measuring Ending
        cal_core= x_measurement*y_measurement;


}
```

ultrasonic.h header file

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <string.h>
#include <stdlib.h>
#include "motors.h"

#define M_PORT PORTB
#define m5 0
#define m6 1
#define m7 2
#define m8 3

int count_x = 0;
static volatile int pulse1 = 0; // integer  to access all though the program

int ultrasonic_1()
{
	DDRD = (DDRD | (1<<0))| (1<<1);  // initializing D0 and D1 for select pins
	DDRD = DDRD | (1<<3);                    // initializing D3 for trigger pin
	DDRD = DDRD & (~(1<<2));                 // initializing D2 for echo pin
	PORTD &= (~(1<<PIND0)); //0
	PORTD &= (~(1<<PIND1));  //0

	_delay_ms(50);

	GICR |= (1<<INT0);    //enabling interrupt 0
	MCUCR|= (1<<ISC00);   //setting interrupt triggering logic change
```

```
	sei();

	for(int j=1;j<10;j++)
	{
		PORTD |= (1<<PIND3);
		_delay_us(15); //triggering the sensor for 15usec
		PORTD &= (~(1<<PIND3));
		count_x =(pulse1/58)+1;
	}
	return count_x;
}

int ultrasonic_2()
{
	DDRD = (DDRD | (1<<0))| (1<<1);  // initializing D0 and D1 for select pins
	DDRD = DDRD | (1<<3);                    // initializing D3 for trigger pin
	DDRD = DDRD & (~(1<<2));                 // initializing D2 for echo pin
	PORTD |= (1<<PIND0);  //1
	PORTD &= (~(1<<PIND1)); //0

	_delay_ms(50);

	GICR |= (1<<INT0);    //enabling interrupt 0
	MCUCR|= (1<<ISC00);   //setting interrupt triggering logic change

	int count_y = 0;

	sei();

	for(int j=1;j<10;j++)
```

```
	{
		PORTD |= (1<<PIND3);
		_delay_us(15); //triggering the sensor for 15usec
		PORTD &= (~(1<<PIND3));
		count_y =(pulse1/58)+1;
	}
	return count_y;
}

int ultrasonic_3()
{
	DDRD = (DDRD | (1<<0))| (1<<1);  // initializing D0 and D1 for select pins
	DDRD = DDRD | (1<<3);                    // initializing D3 for trigger pin
	DDRD = DDRD & (~(1<<2));                 // initializing D2 for echo pin
	PORTD &= (~(1<<PIND0)); //0
	PORTD |= (1<<PIND1);   //1
	_delay_ms(50);

	GICR |= (1<<INT0);    //enabling interrupt 0
	MCUCR|= (1<<ISC00);   //setting interrupt triggering logic change

	int count_z = 0;

	sei();

	while(1)
	{
		//All pins of PORTB as output
		PORTD |= (1<<PIND3);
		_delay_us(15); //triggering the sensor for 15usec
		PORTD &= (~(1<<PIND3));
```

```
            count_z =(pulse1/58)+1;
            int period =100;   //period for stepper motor
            DDRB = 0xFF;
            SM2_PORT = SM2_PORT&~(1<<s6);
            M_PORT = ((((M_PORT&~(1<<m8))&~(1<<m7))&~(1<<m6))&~(1<<m5));

            M_PORT = ((((M_PORT|(1<<m8))&~(1<<m7))&~(1<<m6))|(1<<m5));
            _delay_ms(period);
            M_PORT = ((((M_PORT|(1<<m8))|(1<<m7))&~(1<<m6))&~(1<<m5));
            _delay_ms(period);
            M_PORT = ((((M_PORT&~(1<<m8))|(1<<m7))|(1<<m6))&~(1<<m5));
            _delay_ms(period);
            M_PORT = ((((M_PORT&~(1<<m8))&~(1<<m7))|(1<<m6))|(1<<m5));
            _delay_ms(period);

            if(count_z < 30)
            {
                  break;
            }
            else
            {
                  continue;
            }
      }
      return count_z;
}

ISR(INT0_vect) //interrupt service routine when there is a change in logic level

{
      static volatile int i = 0;
```

```
        if (i==1)//when logic from HIGH to LOW
        {
                TCCR1B=0; //disabling counter
                pulse1=TCNT1;//count memory is updated to integer
                TCNT1=0;//resetting the counter memory
                i=0;
        }
        if (i==0)//when logic change from LOW to HIGH
        {
                TCCR1B|=(1<<CS10) ;//enabling counter
                i=1;
        }
}
```

Name of student: Dinusha K.M (204042N)

1. Responsible part - Bobbin rotation, Cutting unit and Buzzer

Learned to code atmega32 microcontroller using Atmel studio Software and learned to use Proteus software to simulate the circuit. I have responsibilities for the bobbin rotation, cutting unit and buzzer in our automated transform winding project. These are my responsibilities for the project. I was aware of how to get calculate bobbin rotation. For that purpose, I used the stepper motor. this allows us to accurately calculate the number of turns of the bobbin, I used 2 stepper motors for the cutting unit, one of them is to move the cutter here and there and another one to cut the wire using a cutter And I used a buzzer Also, I Learned to code atmega32 microcontroller using Atmel studio Software and learned to use Proteus software to simulate the circuit. I was aware of the working mechanism and pins of all the components that I have used. I did the stepper motor and buzzer coding part and simulated the circuit using proteus software. I drew the schematic diagram and PCB design of those components.

2. Specification

Stepper motor

Every revolution of the stepper motor is divided into a discrete number of steps, in many cases 200 steps, and the motor must be sent a separate pulse for each step. The stepper motor can only take one step at a time and each step is the same size. Since each pulse causes the motor to rotate at a precise angle, typically 1.8°, the motor's position can be controlled without any feedback mechanism. As the digital pulses increase in frequency, the step movement changes into the continuous rotation, with the speed of rotation directly proportional to the frequency of the pulses.

L293D motor driver IC

The L293D IC receives signals from the Atmega32 and transmits the relative signal to the motors. It has two voltage pins, one of which is used to draw current for the working of the L293D and the other is used to apply voltage to the motors. The L293D switches its output signal according to the input received from the Atmega32. For Example: If the Atmega32 sends a

1(digital high) to the Input Pin of L293D, then the L293D transmits a 1(digital high) to the motor from its Output Pin. An important thing to note is that the L293D simply transmits the signal it receives. It does not change the signal in any case. So here one L293D IC has been used to connect a stepper motor with Atmega32.

74HC4051 Demultiplexer

The 74HC4051 is a member of the Industries 74xxx series of Logic devices.it is a single-pole octal-throw analog switch (SP8T) suitable for use in analog or digital 8:1 demultiplexer applications. The switch features three digital select inputs (S0, S1 and S2), eight independent inputs / outputs, a common input/output and a digital enable input .

CD4052 IC

The CD4052 can be used as a 4:1 Multiplexer, that is it can take inputs from 4-channel and convert it to single channel output based on the channel select pins. It is a CMOS logic-based IC belonging to a CD4000 series of integrated circuits. We can use this IC in both digital and analog applications

Buzzer

The buzzer is a sounding device that can convert audio signals into sound signals. It is usually powered by DC voltage. The buzzer is used such as when machining on and off and when the winding process starts.

3. Technique

Bobbin rotation part

When we give the control signal to the stepper motor it will rotate one round. That winds the wire on the bobbin one turn.

Cutting Unit

Here we use a stepper motor to move the cutter. When we need to cut a wire, we activate the stepper motor. Then the cutter moves to the wire through the tread bars. Then it cuts the wire and moves back to the resting position. Here we also use a stepper motor to cut the wire. We have a plier which is used to cut the wire. One hand of the plier is fixed, and another hand is connected to a rotating disk, which is connected to the stepper motor. When we rotate the disk 90° and rotate back to the start position using the stepper motor wire gets cut when the jaws of the plier get extended and retract back.

Buzzer

We use a transistor as a high-speed switch to drive the buzzer. When the "Buzzer" pin is low the transistor act as an open switch. So, there is no current flowing through the buzzer. When we make the "Buzzer" pin high the voltage of the base of the transistor becomes 5V. That changes the transistor to its saturated state, then the current starts flowing from collector to emitter. At this stage transistor act as a closed switch. Since that makes current flow through the buzzer, the buzzer gets activated. Based on the above technique when we generate a pulse on the "Buzzer" pin we can hear a sound based on the frequency of our signal.

4. Schematic Diagram

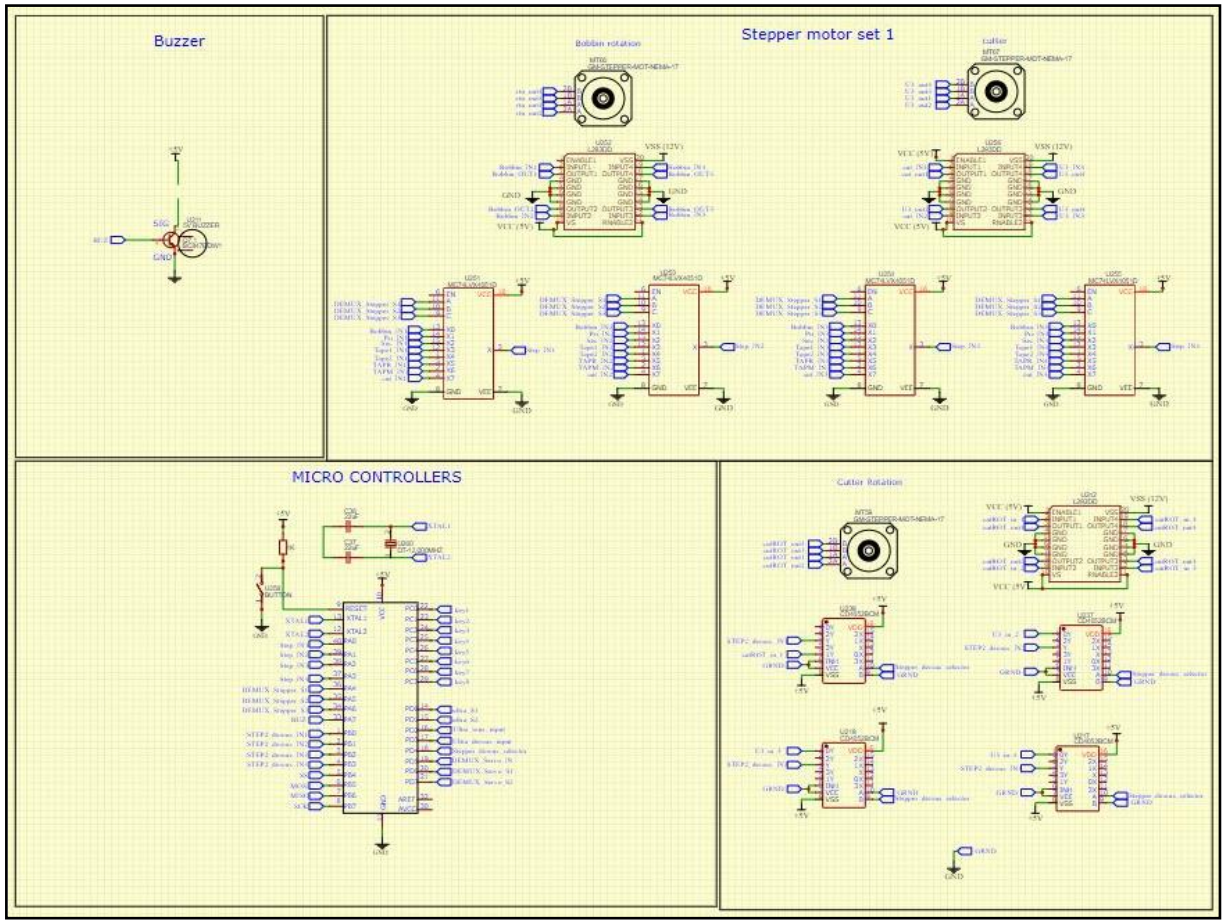

Figure 34

5. PCB Design

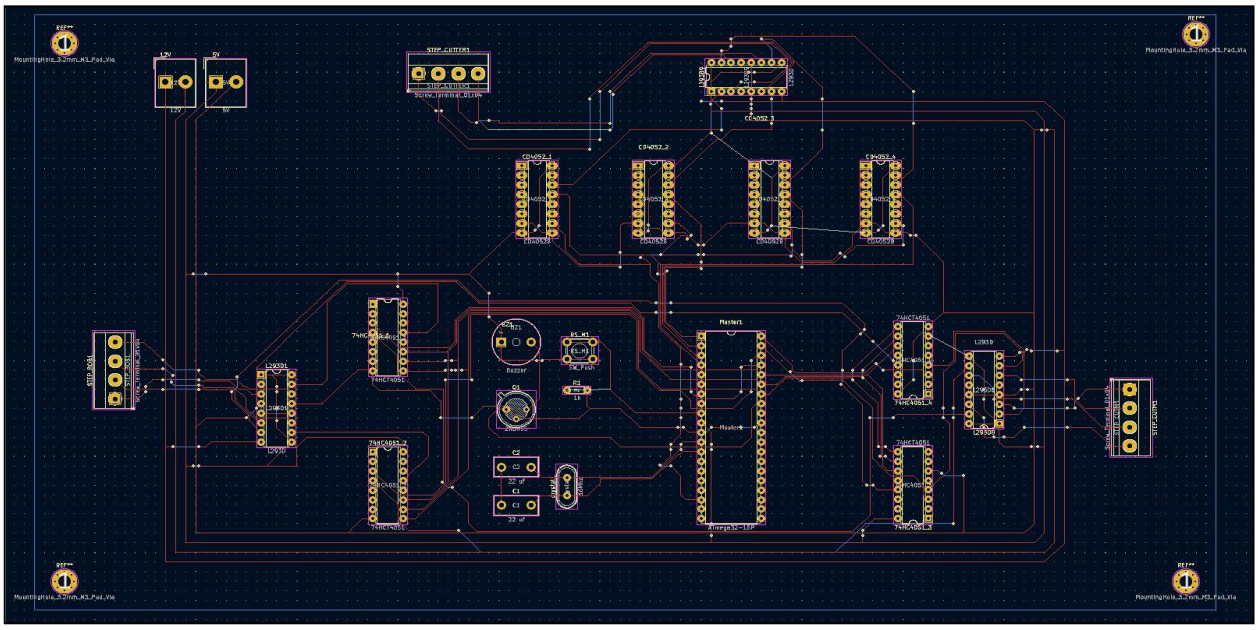

Figure 35

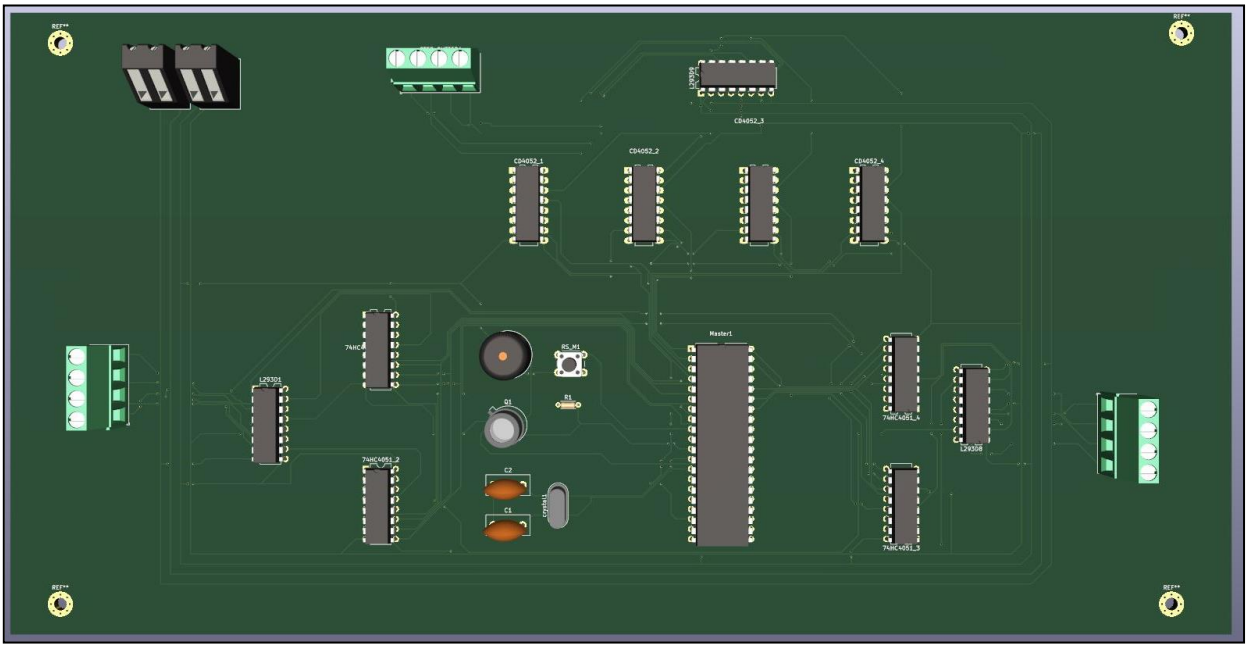

Figure 36

6. Proteus Simulation

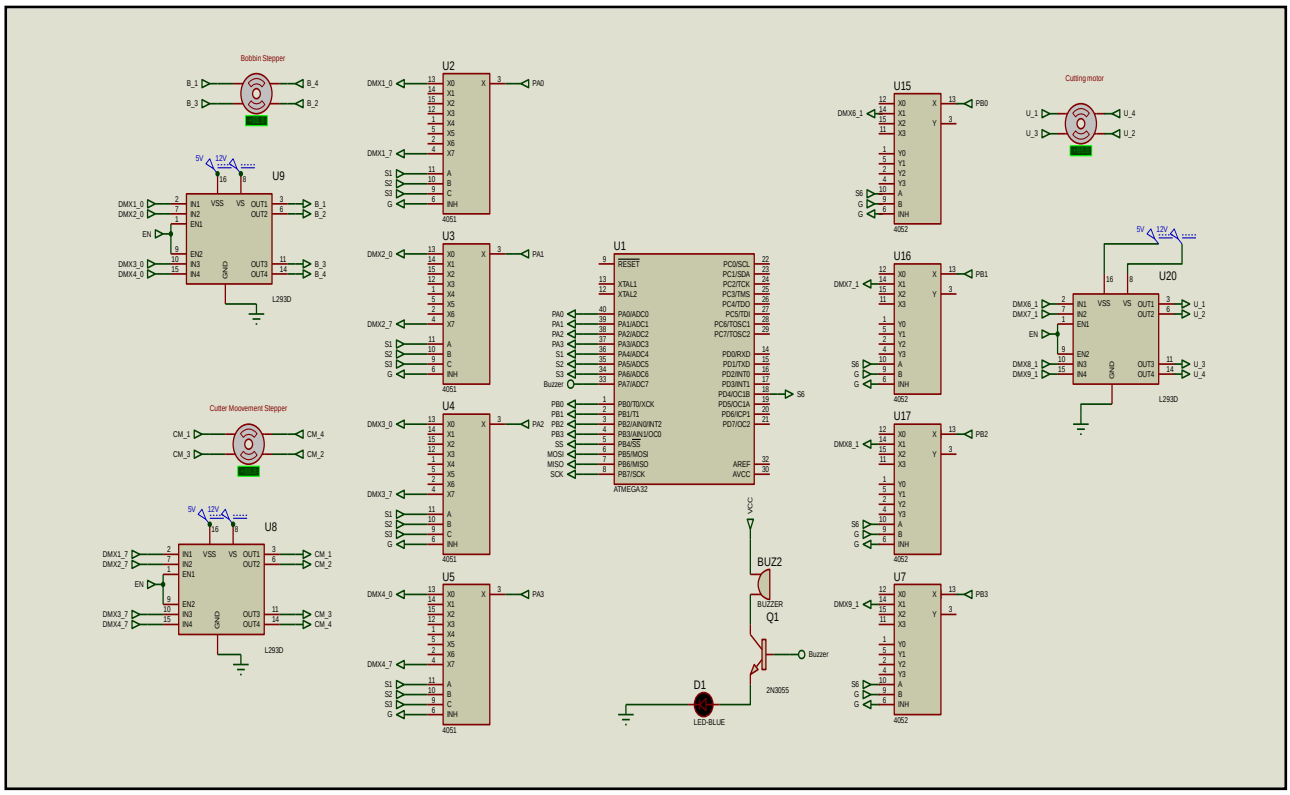

Figure 37

7. Code

```
void bobbin ()
{
SM1_PORT = (((SM1_PORT&~(1<<s3))&~(1<<s2))&~(1<<s1)); // selecting bobbin
stepper
        stepper_1(1);   //      rotate one round of bobbin
}

void cutting_process()
{
        SM1_PORT = (((SM1_PORT|(1<<s3))|(1<<s2))|(1<<s1));
        stepper_1(2);                                   //      moving to pre-defined distance
        SM2_PORT = SM2_PORT|(1<<s6);
        stepper_2();                                    //      moving to cut the wire
        SM1_PORT = (((SM1_PORT|(1<<s3))|(1<<s2))|(1<<s1));
        stepper_1_reverse(2);                           //      moving back to the previous location
}

void stepper_1(int n)
{
        int period;
        period = 1000;
        DDRA = 0xFF;
        while (1)
        {
                M1_PORT = ((((M1_PORT&~(1<<m4))&~(1<<m3))&~(1<<m2))&~(1<<m1));
                for(int i=0;i<n;i++)
                {
                        M1_PORT =((((M1_PORT|(1<<m4))&~(1<<m3))&~(1<<m2))|(1<<m1));
```

```
                _delay_ms(period);
                M1_PORT =((((M1_PORT|(1<<m4))|(1<<m3))&~(1<<m2))&~(1<<m1));
                _delay_ms(period);
                M1_PORT =((((M1_PORT&~(1<<m4))|(1<<m3))|(1<<m2))&~(1<<m1));
                _delay_ms(period);
                M1_PORT =((((M1_PORT&~(1<<m4))&~(1<<m3))|(1<<m2))|(1<<m1));
                _delay_ms(period);
        }
        break;
    }
}

void stepper_2()
{
    int period;
    period = 1000;
    DDRB = 0xFF;
    while (1)
    {
        M2_PORT = ((((M2_PORT&~(1<<m8))&~(1<<m7))&~(1<<m6))&~(1<<m5));
        M2_PORT = ((((M2_PORT|(1<<m8))&~(1<<m7))&~(1<<m6))|(1<<m5));
        _delay_ms(period);
        M2_PORT = ((((M2_PORT&~(1<<m8))&~(1<<m7))|(1<<m6))|(1<<m5));
        _delay_ms(period);
        break;
    }
}
```

Buzzer.c

```
#define F_CPU 8000000UL
```

```
#include <avr/io.h>
#include <util/delay.h>

void buzzer()
{
	DDRA = DDRA | (1<<7);
	for(int i=0;i<2;i++)
	{
		PORTA = PORTA | (1<<7);

		_delay_ms(1000);
		PORTA = PORTA & (~(1<<7));

		_delay_ms(1000);
	}
}
```