### 205108G - THENNAKOON T.M.M. S

**Responsible Part**

* Interfacing and coding Stepper Motor
* Interfacing and coding Keypad
* Designing 3D animation
* Designing PCB

**1.Stepper Motor**

In our project we are using stepper motor to spoon the sap until the kithul treacle or kithul jaggery is made. The handle of spoon is connected to the motor, so the stepper motor is placed above the sap container to mix sap properly.

In our project the stepper motor will activate under following functions.

* When the container is filled with sap, motor stars to rotate and it continues until the temperature set to set temperature
* When the user increases the set temperature after the process is done, motor starts again rotating until temperature set with the new set temperature.

To select a motor, we consider the blow requirements:

* In our project we spoon 4kg sap for each time, so the motor requires to be able to hold that weight.
* We are expecting to produce kithul jaggery, and it is very thick and heavy. For making kithul treacle or jaggery, motor requires high torque.

According to functions and requirements of the project, I choose to use **NEMA 23 (size 57mm) Hybrid Stepper Motor** with **L293D** Driver.This stepper motor is a high torque hybrid stepping motor.



Figure 16 Stepper Motor

Technique and Specification

**NEMA 23 bipolar Stepper Motor**

This bipolar stepper motor has one winding per stator phase. A two-phase bipolar stepper motor will have 4 leads. In a bipolar stepper, we don’t have a common lead like in a unipolar stepper motor.

Diagram, schematic

Description automatically generated

**Electrical Specification**

* Manufacturer Part Number: 23HP22-2804S
* Number Of Phase: 2
* Step Angle: 1.8 deg
* Holding Torque: 1.26 Nm(178.4oz.in)
* Rated Current/phase: 2.8 A
* Phase Resistance: 0.9 ohms± 10%
* Inductance: 2.5 mH ± 20%(1KHz)

**Physical Specification**

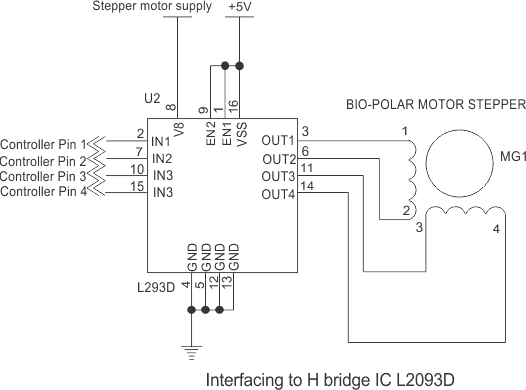
* Frame Size: 57 x 57 mm
* Body Length: 56 mm
* Shaft Diameter: Φ6.35 mm
* Shaft Length: 21 mm
* D-CUT Shaft Length: 15 mm
* Number of Leads: 4
* Lead Length: 300 mm
* Weight: 0.7 kg

To drive a bipolar stepper, we need a driver IC with an internal H bridge circuit. This is because, to reverse the polarity of stator poles, the current needs to be reversed. This can only be done through a H bridge.

There are two other reasons to use an H Bridge IC

1. The current drawn by a stepper motor is quite high. The micro-controller pin can only provide up to 15 mA at maximum. The stepper needs current which is around ten times this value. An external driver IC can handle such high currents.
2. H Bridge is used is that the stator coils are nothing but an inductor. When coil current changes direction a spike is generated. A normal micro-controller pin cannot tolerate such high spikes without damaging itself. Hence to protect micro-controller pins, an H bridge is necessary.

4 micro-controller pins are required to control the motor. We need to provide the L293D with a 5 V supply as well as the voltage at which the motor needs to operate. Since we will be using both the drivers of the IC, we will assert the enable pin for both.



Bipolar Motor Driver Circuit Interfacing Diagram

bipolar stepper motors are a little complex to wire as we have to use a current reversing H bridge driver IC like an L293D.

But the advantage is that the current will flow through the full coil. The resulting torque generated by the motor is larger as compared to a unipolar motor.

**L293D**

**A picture containing electronics, circuit

Description automatically generated**

the drive must be able to supply sufficient current for your stepper. The micro-controller must only provide the step and direction signal to the drive.

The L293D devices is a quadruple high1 current half-H driver. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. This device is solenoid, DC, and bipolar stepping motor, as well as supply application.

Features & Specification:

• Wide Supply-Voltage Range: 4.5 V to 36 V

• Separate Input-Logic Supply

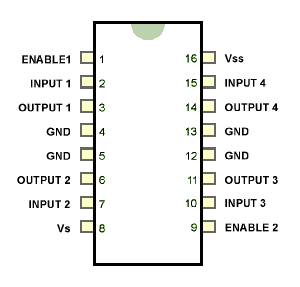
• Internal ESD Protection

• High-Noise-Immunity Inputs

• Output Current 1 A Per Channel (600 mA for designed to drive inductive loads such as relays, L293D)

• Peak Output Current 2 A Per Channel (1.2 A for other high-current/high-voltage loads in positiveL293D)

• Output Clamp Diodes for Inductive Transient Each output is a complete totem-pole drive circuit, Suppression (L293D)

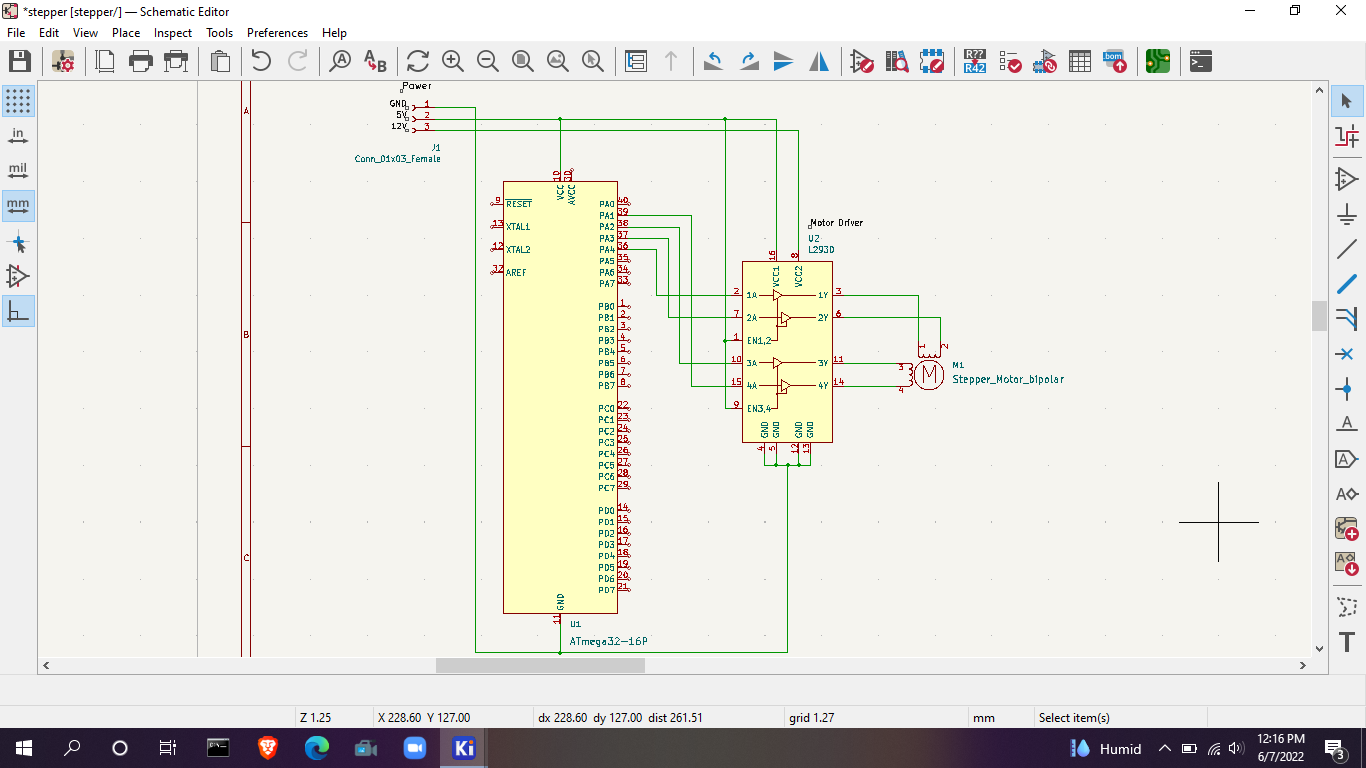


Pin diagram

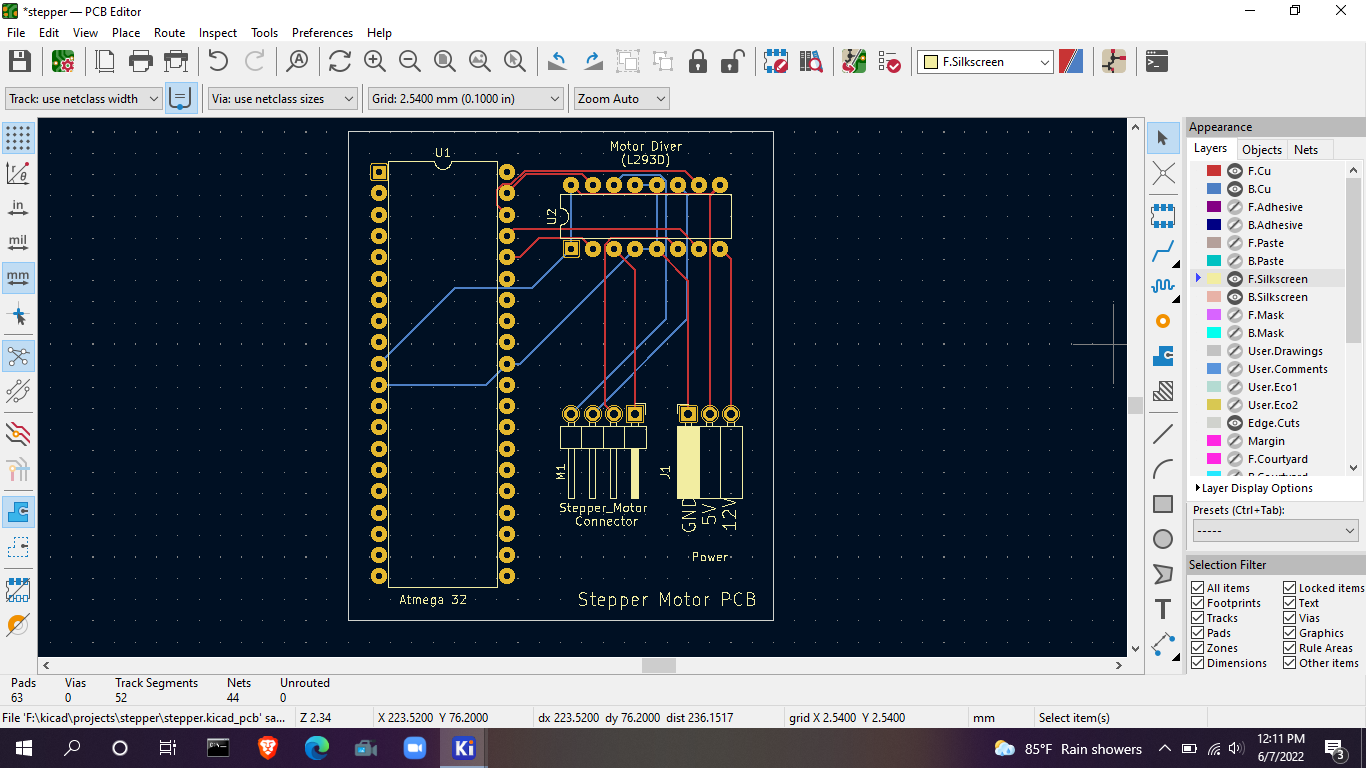
Circuit

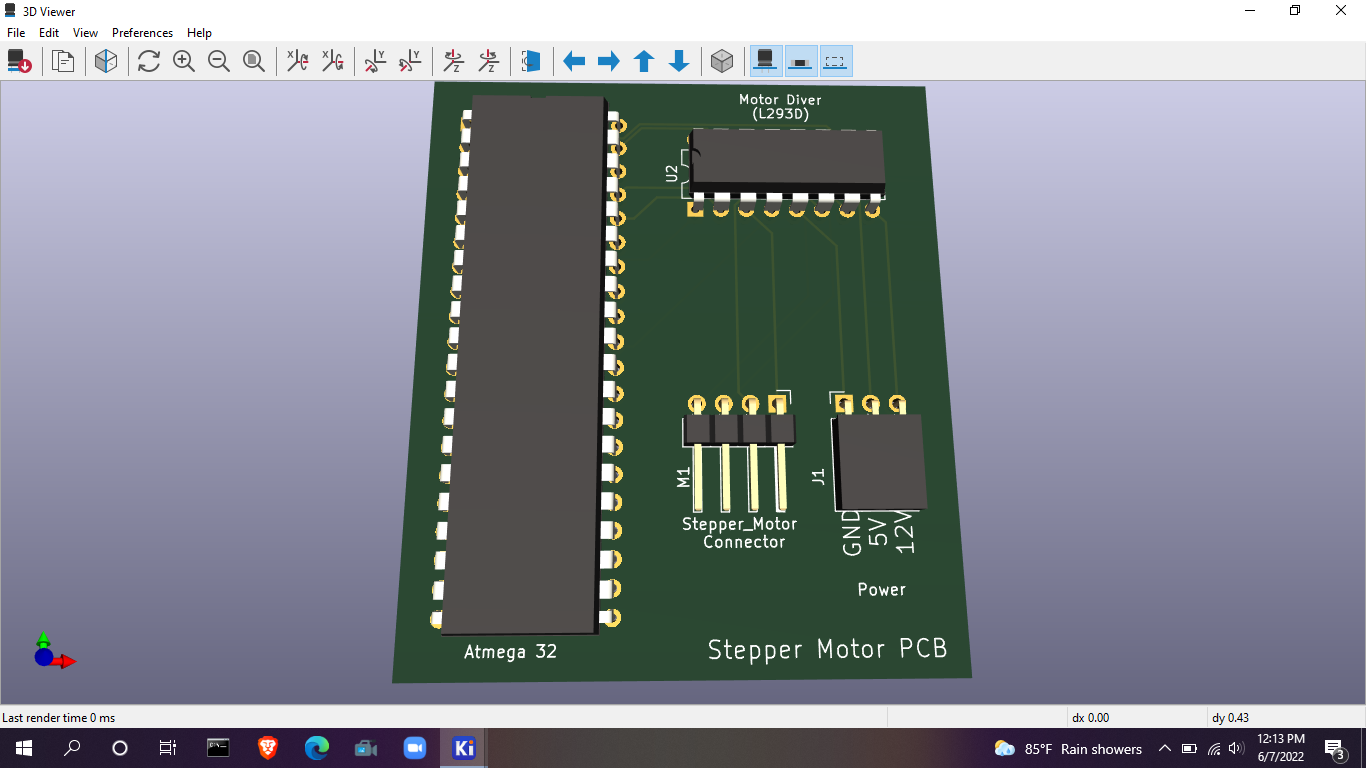
A picture containing schematic

Description automatically generated



PCB design





Code

#define *F\_CPU* 8000000UL

#include <avr/io.h>

#include <util/delay.h>

void stepper();

int main(void)

{

DDRA = 0xFF;

PORTA = 0x00;

/\* Replace with your application code \*/

stepper();

}

void stepper(){

//All pins of PORTC as output //Initially all pins as output high

while(1)

{

PORTA = 0b00010010;

*\_delay\_ms*(100);

PORTA = 0b00000110;

*\_delay\_ms*(100);

PORTA = 0b00001100;

*\_delay\_ms*(100);

PORTA = 0b00011000;

*\_delay\_ms*(100);

}

}

**2. Keypad**

In our project we are using keypad to give options, customize the temperature and open/close valves. we place keypad in front of the container so that it is user friendly.

In our project the keypad will use under following functions.

* Selecting options by keypad makes the user go through the specific process of making kithul treacle or jaggery. When the user selects kithul treacle option or jaggery making option, system sets the defaults (temperature limit, valve no) for the specific product.
* And the user has the feature to increase the default set temperature to continue the process if the user is not satisfied with the output product. User can use 0 to 9 buttons to enter the new temperature.
* At the end, user has the options to open valve or continue the process by using keypad. And after getting the product out, user can close valves by using keypad.

According to functions and requirements of the project, I choose to use **4x4 Matrix Membrane Keypad (#27899)**. This 16-button keypad provides a useful human interface component for microcontroller projects. Convenient adhesive backing provides a simple way to mount the keypad in a variety of applications.

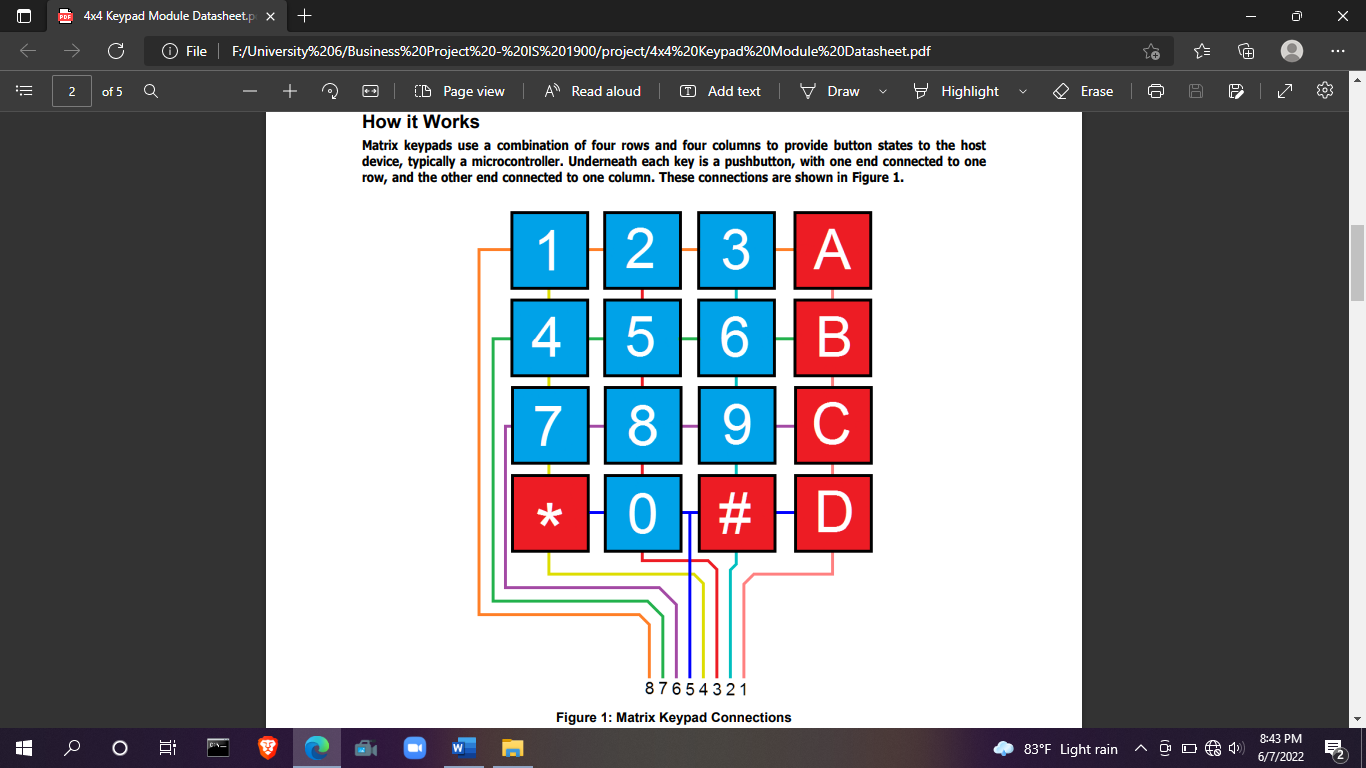
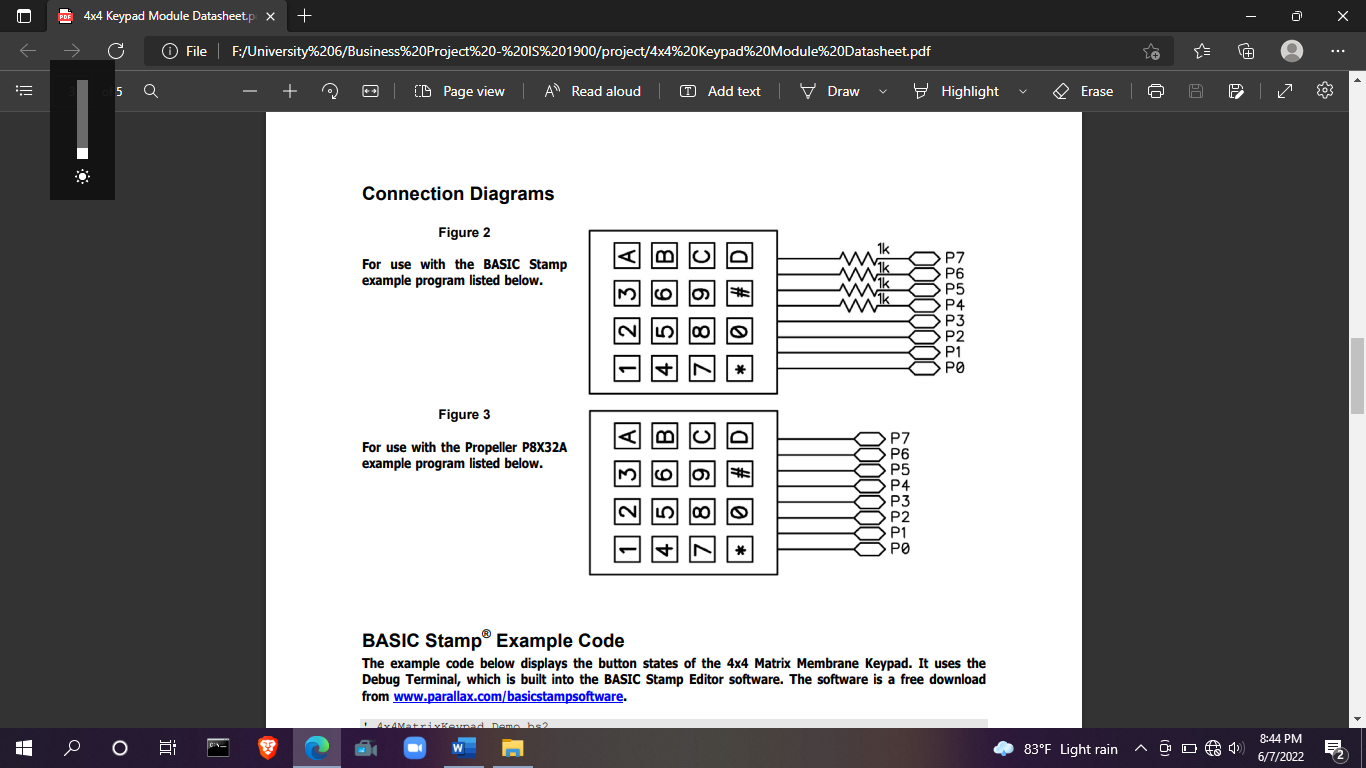


Figure 17 Keypad

Technique and Specification

4\*4 keypad consists of 8 pins that can be identified as 4 input pins and 4 output pins. The interface of the keypad has 16 buttons which are placed between 4 rows and 4 columns. A keypress establishes a connection between the corresponding row and column where the switch is placed. To read the keypress, we need to configure the rows as outputs and columns as inputs. Columns are read after applying signals to the rows in order to determine whether or not a key pressed and if pressed, which key is pressed.

For example, say your program pulls all four columns low and then pulls the first row high. It then reads the input states of each column and reads pin 1 high. This means that a contact has been made between column 4 and row 1, so button ‘A’ has been pressed.

 connection diagram

**Key Features:**

* Ultra-thin design & adhesive backing provides easy integration to any project
* Easy communication with any microcontroller
* Cable included

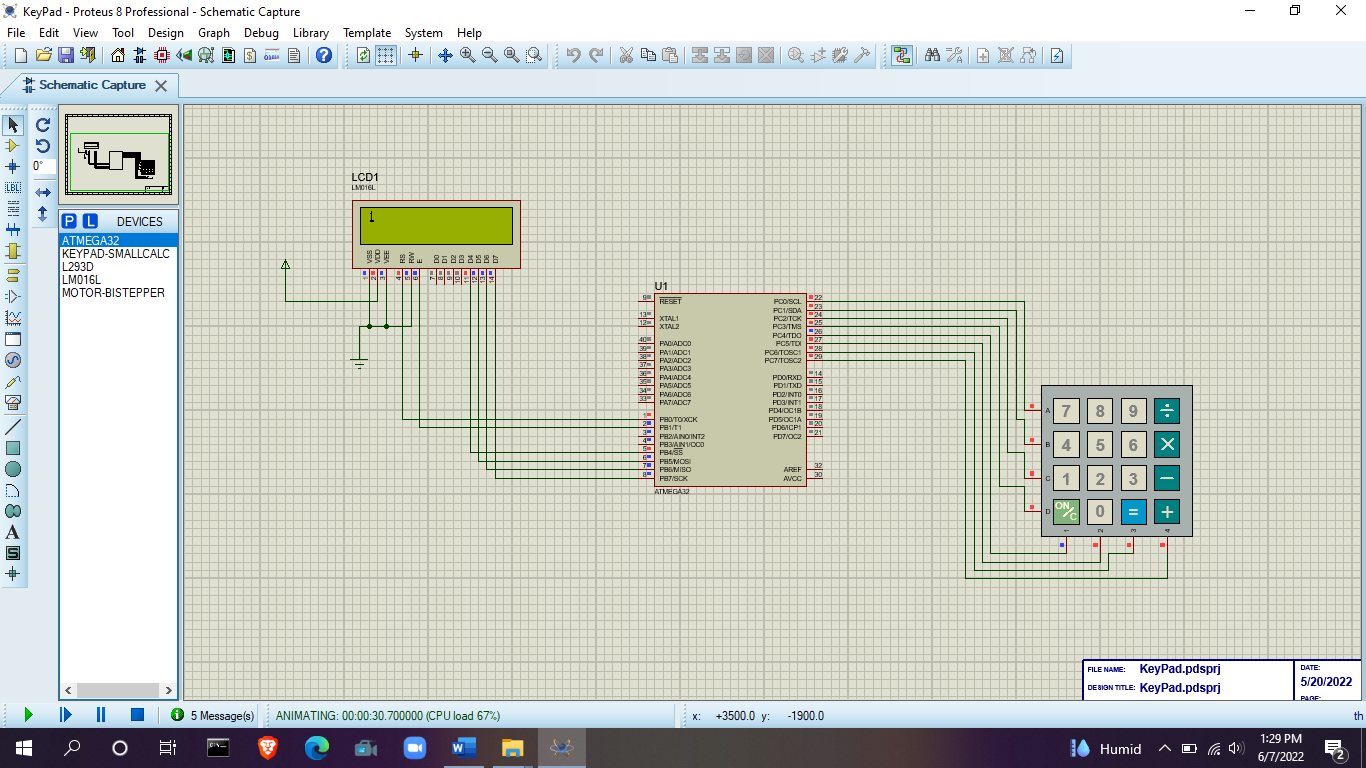
**Details:**

* Maximum Rating: 24 VDC, 30 mA
* Interface: 8-pin access to 4×4 matrix
* Dimensions: Keypad: 2.7 x 3.0 in (6.9 x 7.6 cm)

Cable: 0.78 x 3.5 in (2.0 x 8.8 cm)

* Operating temp range: 32 to 122 °F (0 to 50 °C

Circuit



Keypad circuit design in protues

Schematic

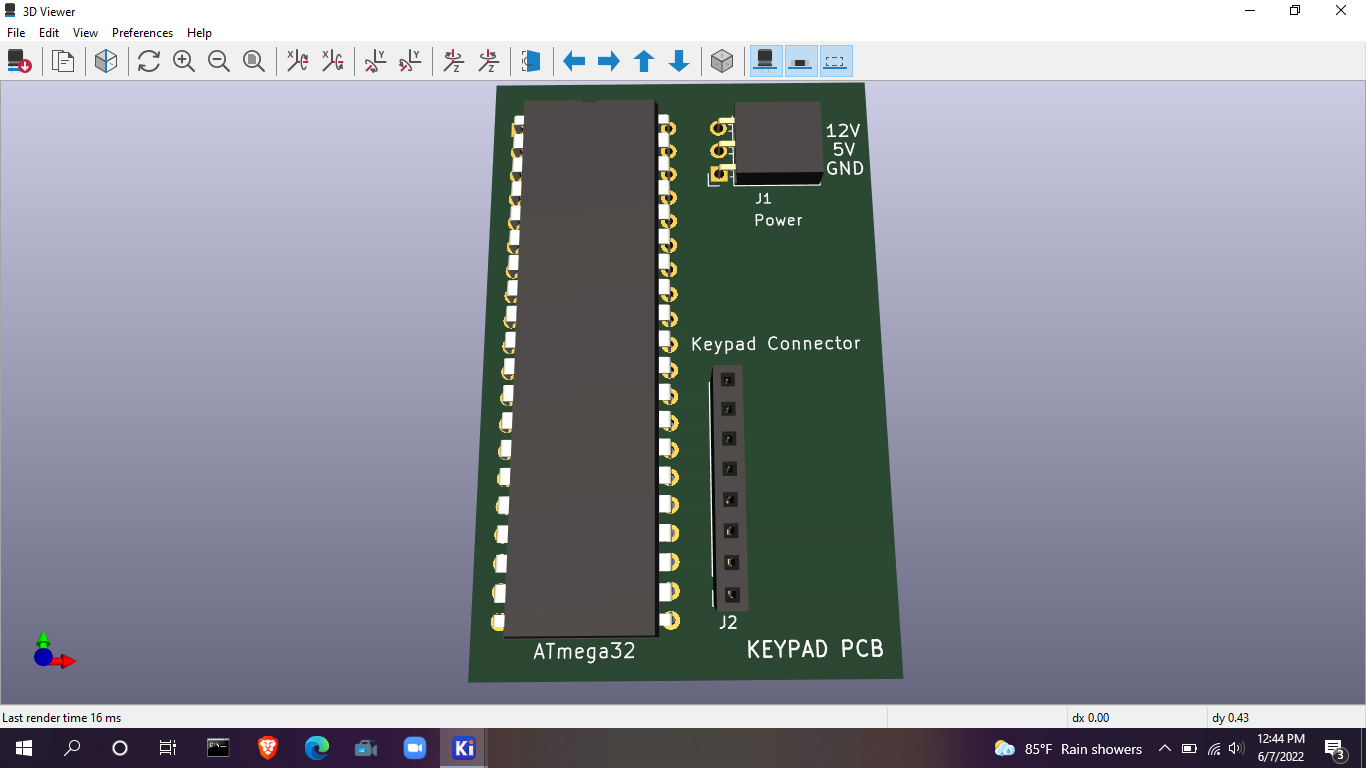
Description automatically generated

Keypad circuit design in kicad

PCB design

A screenshot of a computer

Description automatically generated



3D view of keypad PCB

Code

#define *F\_CPU* 8000000UL

#include <avr/io.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <util/delay.h>

#include <avr/interrupt.h>

////////////////LCD display////////////////

#define LCD\_Dir DDRB /\* Define LCD data port direction \*/

#define LCD\_Port PORTB /\* Define LCD data port \*/

#define RS PB0 /\* Define Register Select (data reg./command reg.) signal pin \*/

#define EN PB1 /\* Define Enable signal pin \*/

/////////////////////////define key pad////////////////////

#define KEY\_PRT PORTC

#define KEY\_DDR DDRC

#define KEY\_PIN PINC

unsigned char keypad[4][4] = {

{'7', '4', '1', 'C'},

{'8', '5', '2', '0'},

{'9', '6', '3', '='},

{'/', 'x', '-', '+'}};

unsigned char colloc, rowloc;

char keyfind();

void LCD\_Command( unsigned char cmnd )

{

LCD\_Port = (LCD\_Port & 0x0F) | (cmnd & 0xF0); /\* sending upper nibble \*/

LCD\_Port &= ~ (1<<RS); /\* RS=0, command reg. \*/

LCD\_Port |= (1<<EN); /\* Enable pulse \*/

*\_delay\_us*(1);

LCD\_Port &= ~ (1<<EN);

*\_delay\_us*(200);

LCD\_Port = (LCD\_Port & 0x0F) | (cmnd << 4); /\* sending lower nibble \*/

LCD\_Port |= (1<<EN);

*\_delay\_us*(1);

LCD\_Port &= ~ (1<<EN);

*\_delay\_ms*(2);

}

void LCD\_Char( unsigned char data )

{

LCD\_Port = (LCD\_Port & 0x0F) | (data & 0xF0); /\* sending upper nibble \*/

LCD\_Port |= (1<<RS); /\* RS=1, data reg. \*/

LCD\_Port|= (1<<EN);

*\_delay\_us*(1);

LCD\_Port &= ~ (1<<EN);

*\_delay\_us*(200);

LCD\_Port = (LCD\_Port & 0x0F) | (data << 4); /\* sending lower nibble \*/

LCD\_Port |= (1<<EN);

*\_delay\_us*(1);

LCD\_Port &= ~ (1<<EN);

*\_delay\_ms*(2);

}

void LCD\_Init (void) /\* LCD Initialize function \*/

{

LCD\_Dir = 0xFF; /\* Make LCD command port direction as o/p \*/

*\_delay\_ms*(20); /\* LCD Power ON delay always >15ms \*/

LCD\_Command(0x33);

LCD\_Command(0x32); /\* send for 4 bit initialization of LCD \*/

LCD\_Command(0x28); /\* Use 2 line and initialize 5\*7 matrix in (4-bit mode)\*/

LCD\_Command(0x0c); /\* Display on cursor off\*/

LCD\_Command(0x06); /\* Increment cursor (shift cursor to right)\*/

LCD\_Command(0x01); /\* Clear display screen\*/

*\_delay\_ms*(2);

LCD\_Command (0x80); /\* Cursor 1st row 0th position \*/

}

void LCD\_String (char \*str) /\* Send string to LCD function \*/

{

int i;

for(i=0;str[i]!=0;i++) /\* Send each char of string till the NULL \*/

{

LCD\_Char (str[i]);

}

}

void LCD\_String\_xy (char row, char pos, char \*str) /\* Send string to LCD with xy position \*/

{

if (row == 0 && pos<16)

LCD\_Command((pos & 0x0F)|0x80); /\* Command of first row and required position<16 \*/

else if (row == 1 && pos<16)

LCD\_Command((pos & 0x0F)|0xC0); /\* Command of first row and required position<16 \*/

LCD\_String(str); /\* Call LCD string function \*/

}

void LCD\_Clear()

{

LCD\_Command (0x01); /\* Clear display \*/

*\_delay\_ms*(2);

LCD\_Command (0x80); /\* Cursor 1st row 0th position \*/

}

int main(void)

{

LCD\_Init();

LCD\_Command(0xc1);

LCD\_String("welcome");

*\_delay\_ms*(1000);

LCD\_Clear();

while (1)

{

char a = keyfind();

LCD\_Char(a);

*\_delay\_ms*(1000);

LCD\_Clear();

LCD\_Clear();

}

}

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]);

}