Department of Electronic and Telecommunication Engineering

University of Moratuwa

EN2160 - Electronic Design Realization

Automatic Batch Code Printer

Design Details Report

Anjula M.K 210368V

Meemanage N.A 210385U

Contents

I. Comprehensive Design Details

A. Introduction to the proposed printing process.

The process of batch code printing involves several intricate steps orchestrated by multiple ATmega chips to ensure efficient and accurate printing on various surfaces. Here's a refined description of the procedure:

The printing process begins with an ATmega chip generating bit patterns (matrices) for alphanumeric characters. Each character input by the user is translated into a column-by-column bit pattern, forming a serial signal for each column. These serial signals, representing the columns of the characters, are transmitted to another ATmega chip through shift registers, where they are converted into voltage signals.

Shift registers play a pivotal role in converting the serial signals into parallel form, facilitating efficient processing and manipulation of the data. They sequentially accept individual bits of the serial data stream and present them simultaneously on multiple output lines in parallel form.

Subsequently, the parallel lines created by the shift registers receive high PWM signals at points corresponding to the highest bits in each column, while lower PWM signals are supplied to other points in the column. Conditional statements are employed to prevent overlapping during printing. If overlap is detected with the previous column, printing is halted; otherwise, PWM signals are supplied to print the column.

Additionally, the first ATmega chip incorporates special functions, such as controlling a proximity sensor to detect the presence of a printing surface near the print head. Another ATmega chip is responsible for regulating the pressure in the ink tank. It adjusts PWM signals based on pressure levels, ensuring optimal printing conditions. Moreover, an analog voltage signal is sent to the first ATmega chip to monitor pressure levels in the tank. If the pressure falls outside the acceptable range, the printing process is aborted until the pressure is corrected.

In summary, the printing process involves meticulous coordination between multiple ATmega chips to generate precise bit patterns, convert serial signals into voltage signals, adjust PWM signals for printing, and monitor printing conditions to ensure high-quality output on various surfaces.

Advantages

The piezoelectric method of batch code printing, as described, offers several advantages over thermal inkjet and other types of batch code printing, such as using pressure valves. Here are some of the key advantages:

Precision and Quality

- Higher Resolution**:** The piezoelectric method can produce finer droplets and more precise control over droplet size and placement, leading to higher resolution prints.
- Accurate Bit Patterns**:** The use of ATmega chips and shift registers ensures that the generated bit patterns for alphanumeric characters are precise, reducing errors and enhancing the clarity of printed codes.

Efficiency and Speed

- Fast Processing**:** The parallel processing capabilities of shift registers allow for efficient handling of serial data streams, speeding up the printing process.

- Non-Overlapping Printing**:** Conditional statements to prevent overlapping ensure that printing is smooth and uninterrupted, which enhances overall printing speed and efficiency.

Versatility

- Multiple Surfaces**:** The proximity sensor integration allows the printer to detect various printing surfaces, making it adaptable to different materials and shapes.
- Variable Print Conditions: The system can adjust PWM signals based on real-time conditions such as pressure levels in the ink tank, ensuring optimal printing on a wide range of surfaces and environments.

Durability and Maintenance

- Less Heat Generation**:** Unlike thermal inkjet printers, which rely on heat to create ink droplets, the piezoelectric method generates less heat, reducing wear and tear on components and potentially extending the printer's lifespan.
- Reduced Clogging: The absence of heat in the droplet formation process minimizes the risk of ink clogging the nozzles, leading to lower maintenance requirements.

Control and Customization

- Precise Control of Droplet Formation: The ability to control droplet size and frequency through high PWM signals allows for more customized printing outputs.
- Dynamic Adjustment: Real-time monitoring and adjustment of printing parameters (such as pressure in the ink tank) ensure consistent print quality and reduce the likelihood of errors.

Environmental Considerations

- Lower Energy Consumption: Since the piezoelectric method does not rely on heating elements, it generally consumes less energy compared to thermal inkjet printers.
- Less Waste: The precise control over ink droplet formation leads to less ink wastage, making the process more environmentally friendly.

Reliability

- Robust Mechanisms: The integration of multiple ATmega chips to handle various tasks (such as bit pattern generation, proximity sensing, and pressure regulation) ensures a robust and reliable printing process.
- Fail-Safe Operations: Conditional checks and real-time monitoring (e.g., pressure levels) provide fail-safe mechanisms to prevent printing errors and ensure high-quality output.

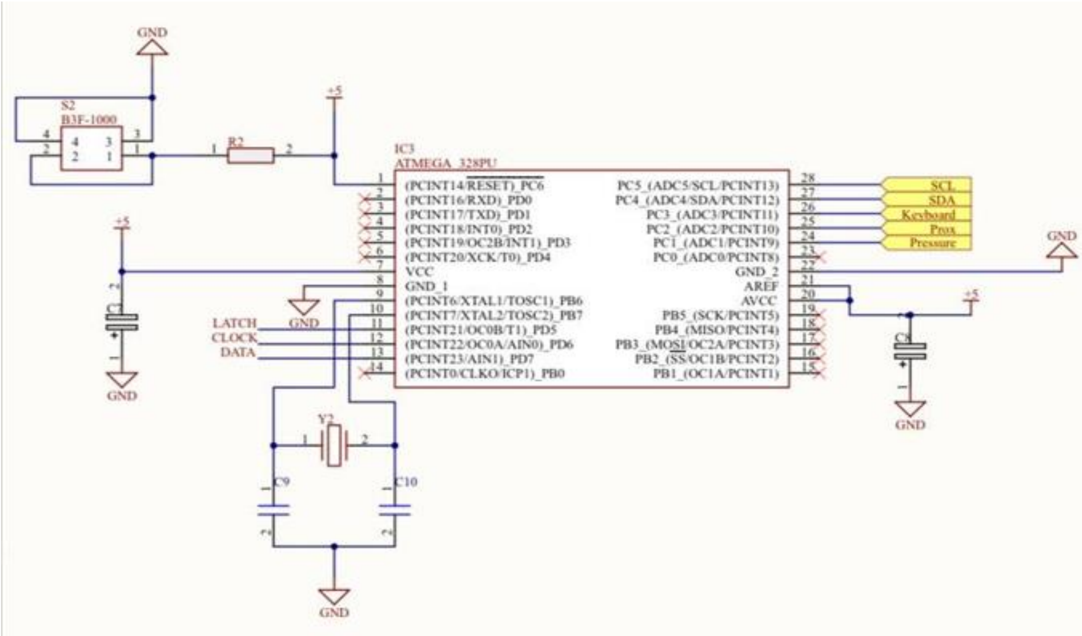
B. Detailed Functionality of the circuit

This module is an integral component designed for taking user input via a keyboard and a display, subsequently converting the input into a printable format for batch code printing. It is specifically developed to ensure precise and reliable printing of batch codes, which are crucial for product identification and tracking in various industries. The detailed functionalities, including text input processing, bitmap mapping, signal generation, and additional control features, are outlined below:

1. **User Input and Display Interface**: The circuit is designed to take user input through a keyboard and display the input on a screen. This allows the user to enter the desired text for batch codes and verify it before printing.
2. **ASCII to Bitmap Conversion**: Once the text is entered, a program maps the ASCII characters of the text into a bitmap format suitable for dot matrix printing. This conversion ensures that each character is accurately represented as a series of dots.
3. **Column Activation Signal Generation**: The module determines which positions of the dot matrix columns should be activated based on the bitmap representation. This information is then converted into a serial signal.
4. **Shift Register Communication**: The serial signal containing the column activation information is sent to a shift register. The shift register processes this serial signal and outputs it to another microcontroller.
5. **Microcontroller and PWM Signal Creation**: The secondary microcontroller receives the column activation information from the shift register. It then checks which positions are activated and generates a corresponding PWM (Pulse Width Modulation) signal as the output.
6. **Proximity Sensor Integration**: The module includes a proximity sensor to detect whether an item is nearby. This ensures that the printing process is initiated only when the item is in the correct position, preventing misprints and ensuring accuracy.
7. **Ink Pressure Monitoring and Control**: Another microcontroller is dedicated to checking the ink pressure and controlling the ink flow. This ensures consistent and high-quality printing by maintaining optimal ink pressure throughout the printing process.

By integrating user input processing, precise bitmap mapping, reliable signal generation, proximity detection, and ink flow control, this module provides a comprehensive solution for batch code printing. These features are essential for product tracking and quality control in various applications, ensuring precision and reliability in the printing process.

1. Main microcontroller.



The main microcontroller serves as the central component of our system it is programmed by Microchip studio platform. It performe crucial tasks to ensure seamless operation. It processes user input from a keyboard and display, allowing users to input and verify text for batch code printing. This text is then converted into a bitmap format, translating each character into a dot matrix representation essential for accurate printing. The microcontroller generates a serial signal from this bitmap data to indicate which dot matrix columns should be activated, sending this signal to a shift register for further processing.

In addition to managing text input and conversion, the main microcontroller integrates with a proximity sensor to detect the presence of an item, ensuring that printing occurs only when the item is correctly positioned. This prevents misprints and enhances accuracy. It also communicates with a secondary microcontroller responsible for monitoring ink pressure and controlling ink flow, maintaining consistent and high-quality printing by adjusting the ink pressure as needed.

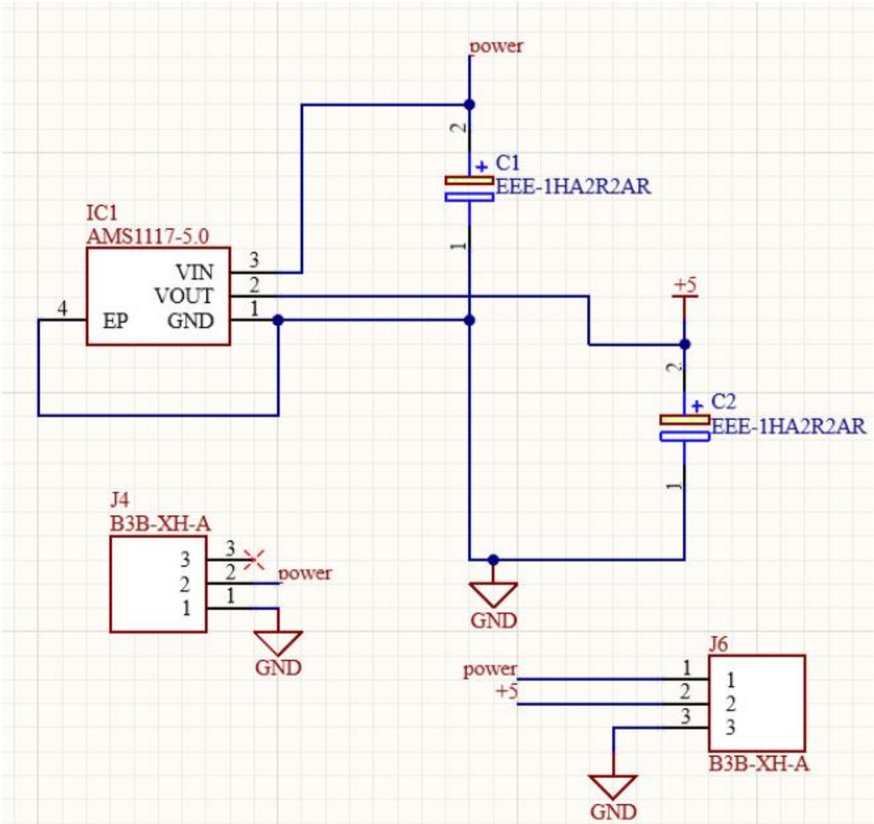
By LATCH, CLOCK and DATA pin it sends data to shift register to decode from there. SCL and SDA pins are used to initiate the OLED Display. Keyboard is connected to analog pin 3 and it initiate proper action based on the analog voltage value is received. Proximity sensor is connected to Analog pin 2. The pressure measurement value from the pressure controller is received by Analog pin 1.

By coordinating these tasks and managing communication between various system components, the main microcontroller ensures the overall efficiency and reliability of the batch code printing system, essential for accurate product identification and tracking.

Component list of power supply circuit

item	quantity	reference	part
1	1	IC3	8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash
2	1	S2	Tactile Switches 6X6 Flat 4.3mm Btn Force 100g w/o Grd
3	1	Y2	Crystals CRYSTAL 16.1280MHZ 18PF T/H
4	1	R2	Thick Film Resistors - SMD 100 OHM 1% 1/2W
5	2	C7,C8	Aluminum Electrolytic Capacitors - SMD 25V 10uF 20%
6	2	C9 , C10	Multilayer Ceramic Capacitors MLCC - SMD/SMT 25V 2.2uF X7R 0805 1 0% AEC-Q200 FLEXITER

2. Power supply circuit.



This power supply module is designed to provide a stable 2A maximum current output using the AMS1117 regulator IC, ensuring consistent performance and reliability for various ICs and components on a PCB. The AMS1117 regulator offers high-precision fixed voltage regulation and can handle up to 2A of current, making it suitable for a wide range of applications, from small electronic devices to larger systems. It accepts a broad input voltage range, typically from 4.5V to 15V, allowing compatibility with various power sources such as batteries, wall adapters, and other external supplies. The regulator includes built-in thermal protection to prevent overheating and short-circuit protection to safeguard against accidental shorts, enhancing the system's overall reliability. With a low dropout voltage of approximately 1.1V at 2A, the AMS1117 ensures efficiency even when the input voltage is close to the output voltage. The compact design of the power supply module allows for easy integration into space-constrained applications. Additionally, the inclusion of input and output capacitors for filtering helps smooth out voltage fluctuations and reduce noise, providing a clean and stable power supply for sensitive electronic components. This comprehensive design makes the power supply module an efficient and reliable solution for powering various ICs and components in electronic applications.

By pass capasitors have been added to input and output pins to filterout any high frequency noices that can effect the smooth operation of ICs.

Component list of power supply circuit

item	quantity	reference	part
1	1	IC1	1A LOW DROPOUT VOLTAGE REGULATOR
2	2	C1 , C2	Aluminum Electrolytic Capacitors - SMD 2.2UF 50V VS SMD
3	2	J4 , J6	Connector Header Through Hole 3 position 0.098" (2.50mm)

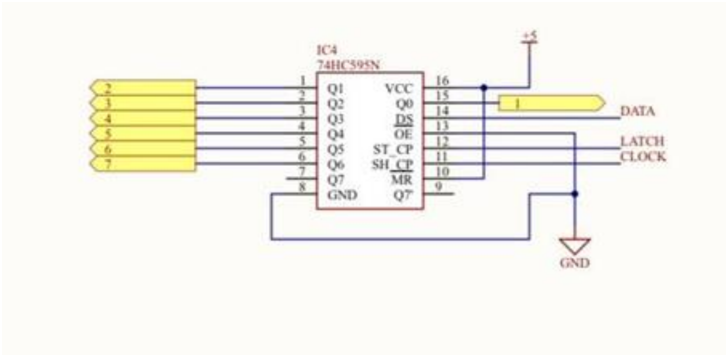
3. Proximity Sensor.

The proximity sensor that have been used is FA-CDD-40N .The CDD-40N photoelectric sensor is a diffuse reflection sensor designed for precise object detection within a range of up to 40 cm. It operates by emitting an infrared light beam that reflects off an object's surface and detecting the reflected light. The sensor features an NPN open drain output, capable of sinking current up to 100mA with a maximum input voltage of 30VDC, making it suitable for interfacing with various control systems. It operates on a power supply range of 10-30 VDC, with a 10% ripple tolerance for stable performance. The sensor includes a 2-meter cable for flexible installation and has compact dimensions of 69 mm in length and 18 mm in diameter. It functions effectively within a temperature range of -25 to 55 degrees Celsius and boasts a fast response time of 1.5 milliseconds. Utilizing an infrared light source, it provides reliable performance in diverse lighting conditions with minimal interference. With a maximum lag phase of 10%, the CDD-40N ensures quick detection and signal output, making it ideal for applications in industrial automation, packaging, safety systems, robotics, and material handling.

4. Keyboard

The keyboard that have been used is MD0414 it inputs 5 v and current of 100mA. It out puts Analog voltage based on the key that have been pressed. It have 5 keys , 4 arrow keys for direction and an enter key. We can display the characters on the key board and user can use the key board to choose the combination of the cracters to input for printing.

5. Shift Registers

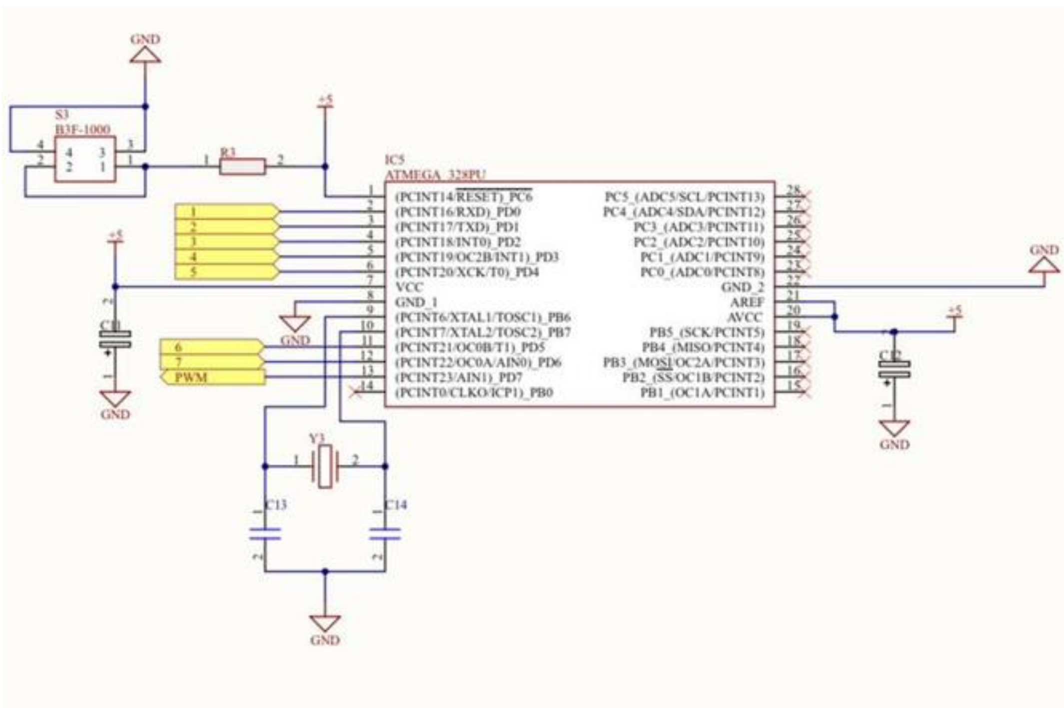


The shift register is a crucial component in the batch code printing system, responsible for converting serial signals into parallel signals to precisely control the printing mechanism. Its primary functionalities include serial-to-parallel conversion, data storage, shifting, and parallel output.

Firstly, the shift register receives serial data from the main microcontroller latch and clock pins syncronise the data stream and data pin receives the serial data specifying which columns of the dot matrix should be activated for printing. This serial data is then stored in the shift register's internal registers. As new bits of serial data are received, the existing bits are sequentially shifted to ensure proper alignment and sequencing.

Once all serial data has been received and stored, the shift register converts it into parallel signals. Each bit in the parallel output corresponds to the state (activated or deactivated) of a specific column in the dot matrix. This parallel output is crucial for enabling precise control over the printing process, ensuring accurate placement of dots to form batch codes.

6. Secondary Microcontroller



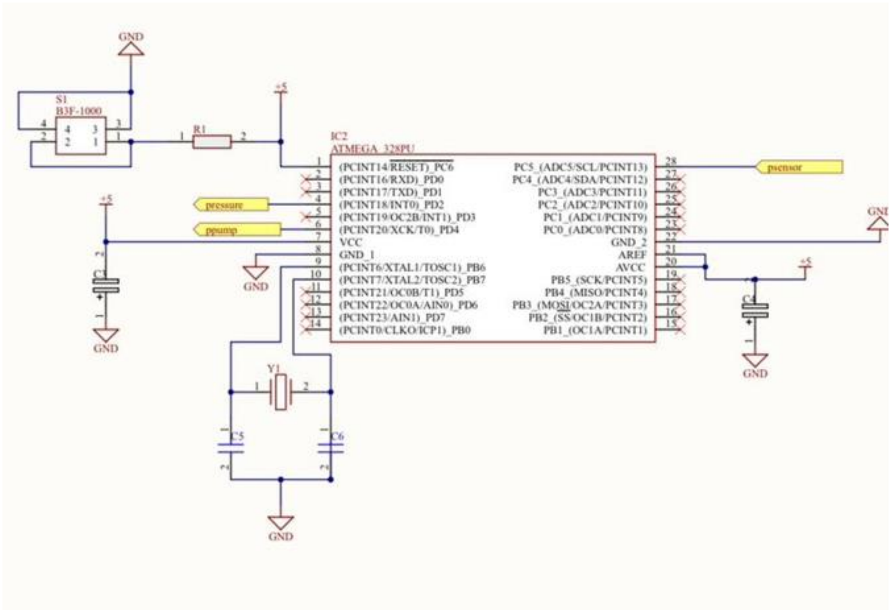
The secondary microcontroller enhances the batch code printing process by generating Pulse Width Modulation (PWM) signals based on the activated dot matrix columns. Its primary functionality lies in PWM signal generation and coordination with the main microcontroller.It takes sense inputs from digital pins 0 to 6 and output the PWM in Digital pin 7.

Upon receiving parallel signals from the shift register, the secondary microcontroller interprets which dot matrix columns are activated. It then creates PWM signals based on this information, determining the timing and intensity of dot placements on the item being printed.

Component list of power supply circuit

item	quantity	reference	part
1	1	IC5	8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash
2	1	S3	Tactile Switches 6X6 Flat 4.3mm Btn Force 100g w/o Grd
3	1	Y3	Crystals CRYSTAL 16.1280MHZ 18PF T/H
4	1	R3	Thick Film Resistors - SMD 100 OHM 1% 1/2W
5	2	C11,C12	Aluminum Electrolytic Capacitors - SMD 25V 10uF 20%
6	2	C13 , C14	Multilayer Ceramic Capacitors MLCC - SMD/SMT 25V 2.2uF X7R 0805 1 0% AEC-Q200 FLEXITER

7. Pressure controlling system.



The pressure monitoring microcontroller is a critical component in our batch code printing system, dedicated to maintaining optimal ink pressure for consistent and high-quality printing. It reads pressure values from a sensor within the ink supply system and continuously monitors them to ensure they remain within the optimal range. Based on these readings, the microcontroller generates Pulse Width Modulation (PWM) signals to control the speed and flow of ink to the printing mechanism. This precise control prevents issues such as over-inking or under-inking, ensuring consistent print quality. The pressure monitoring microcontroller communicates with the main microcontroller, providing real-time pressure data. The main microcontroller evaluates these readings against predefined thresholds and can halt the printing process if the pressure exceeds or falls below the optimal range, thereby preventing printing defects. By maintaining stable ink pressure, the pressure monitoring microcontroller enhances the reliability and efficiency of the batch code printing system, crucial for accurate product identification and traceability in industrial applications. It takes Input from the sensor to Digital pin 2 and gives the PWM to pressure control curicuit through Digital pin 4 and it gives scaler voltage of the pressure to main controller through Analog pin 5

Components of Pressure control circuit

item	quantity	reference	part
1	1	IC2	8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash
2	1	S1	Tactile Switches 6X6 Flat 4.3mm Btn Force 100g w/o Grd
3	1	Y1	Crystals CRYSTAL 16.1280MHZ 18PF T/H
4	1	R1	Thick Film Resistors - SMD 100 OHM 1% 1/2W
5	2	C3,C4	Aluminum Electrolytic Capacitors - SMD 25V 10uF 20%
6	2	C5 , C6	Multilayer Ceramic Capacitors MLCC - SMD/SMT 25V 2.2uF X7R 0805 1 0% AEC-Q200 FLEXITER

8. Pressure sensor.

The WNK811 pressure sensor that have been used is designed to measure pressures within a range of -1 to 60 MPa with high accuracy, ensuring precise monitoring and control in various applications. Manufactured by WNK and certified with CE, it offers an accuracy of 1% FS and supports multiple output signals including 4-20mA, 0.5-4.5V, i2c, and others, providing flexibility for integration into different systems. The sensor operates effectively within a wide temperature range of -40 to 85°C and can measure pressures up to 10 bar. It is powered by various DC voltages (24VDC, 12VDC, 5VDC, 8-30VDC), enhancing its versatility. Constructed from durable materials such as 304 stainless steel and 316L, it is suitable for use with a range of media including water, liquids, gases, steam, oil, and lubricants. The sensor features various pressure port options (G1/4, G1/2, 1/8NPT, 1/4NPT, and others) to accommodate different connection requirements. Additionally, it can withstand overload pressures up to twice its full scale (FS) and proof pressures up to three times FS, ensuring robust performance in demanding environments.

9. Display

The OLED display operates efficiently across a voltage range of 3.3V to 5V .It offer a vibrant visual experience with its 128 × 64 pixel resolution and 0.15 × 0.15 mm pixel size. It is supported by the SSD1315 driver to integrates seamlessly via 4-wire SPI or I2C interfaces ensuring compatibility with various microcontrollers. The 26.00 × 26.00 mm module size provides a compact yet detailed viewing area. It is available in configurations such as upper yellow & lower blue (C), white (D) and blue (E and it caters to diverse aesthetic preferences and application needs making it suitable for compact electronics requiring high-resolution graphical displays.

II. PCB Design and 3D view

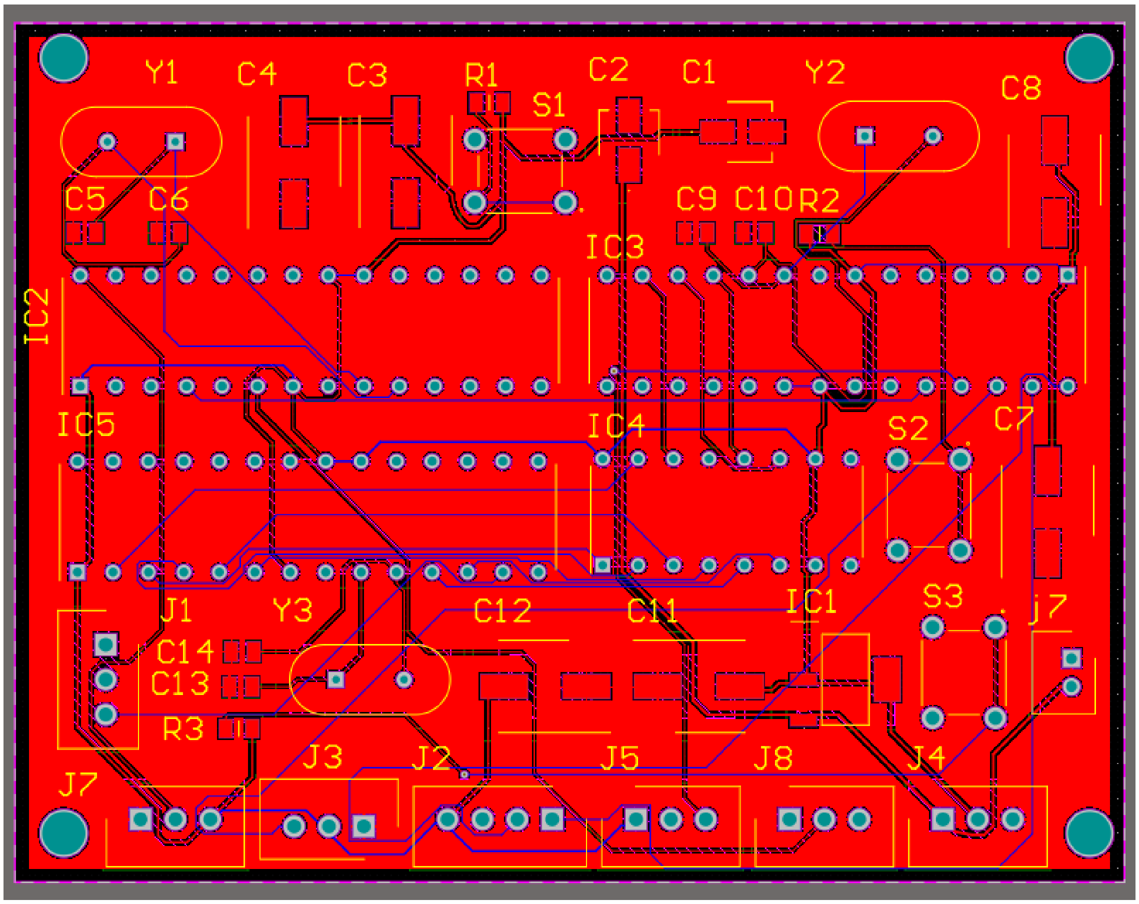


Figure 1

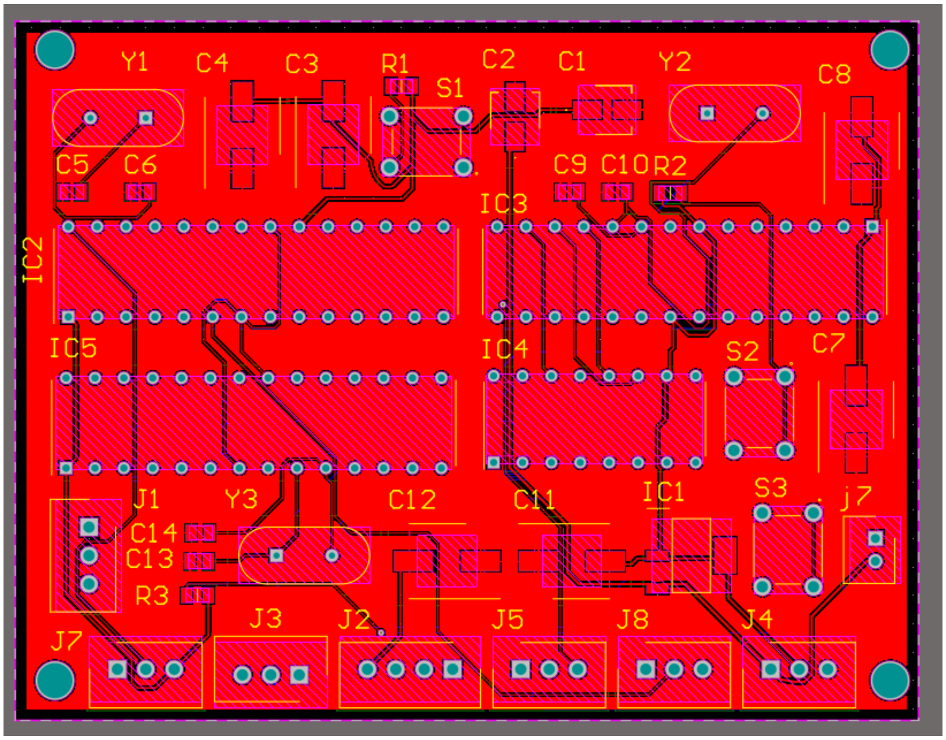


Figure 2

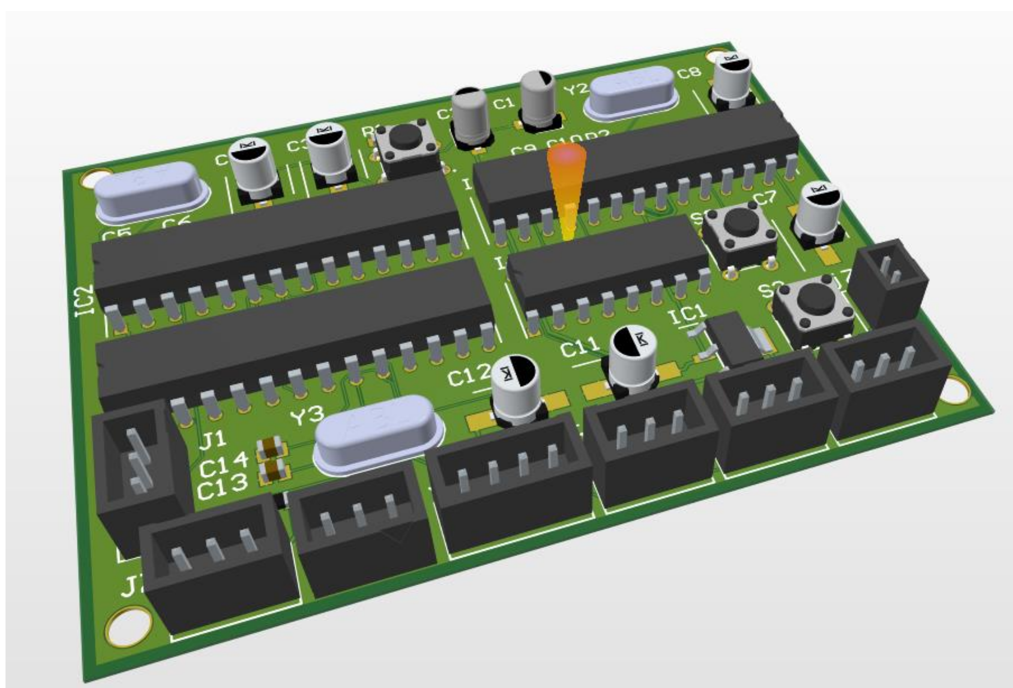
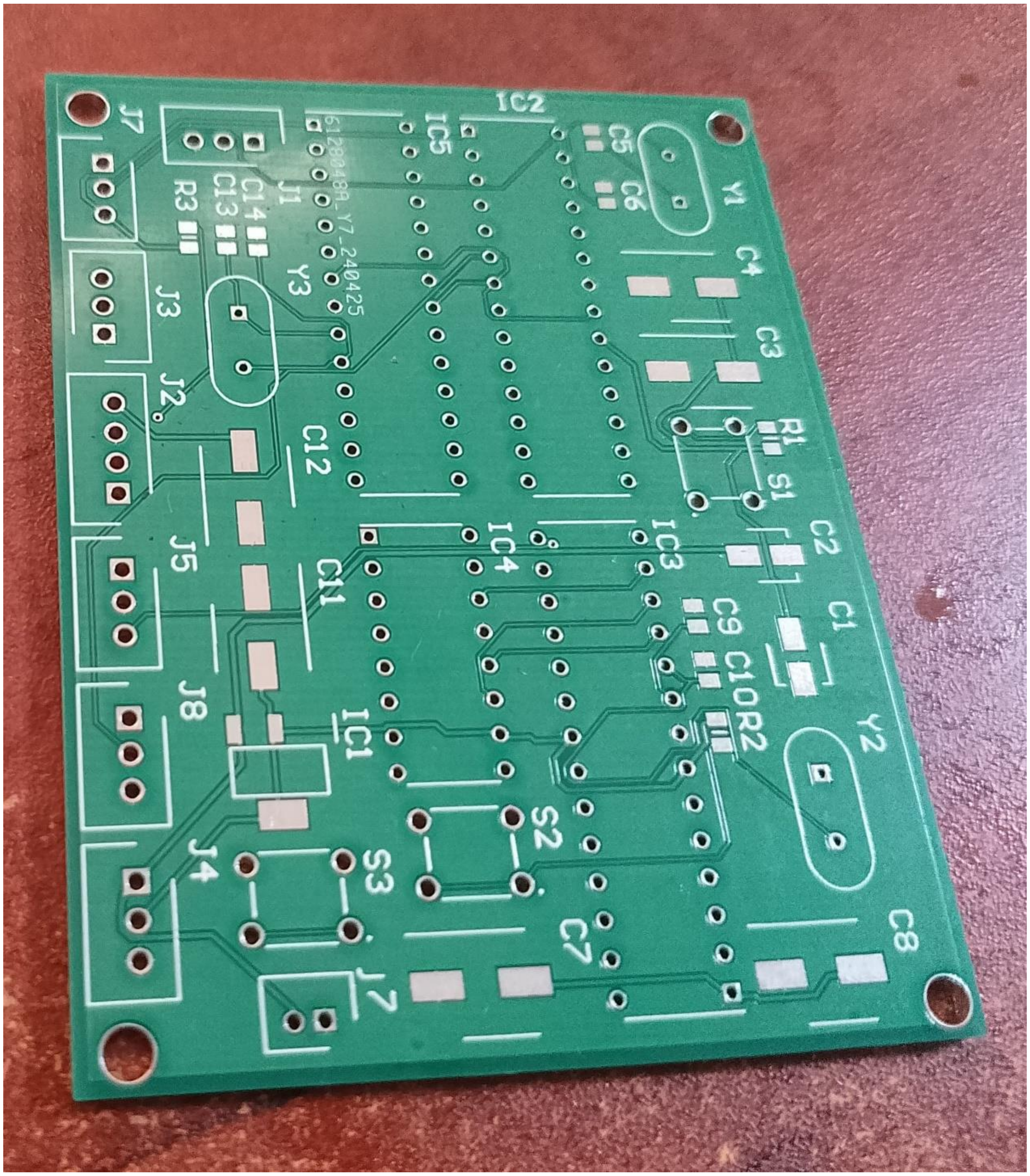
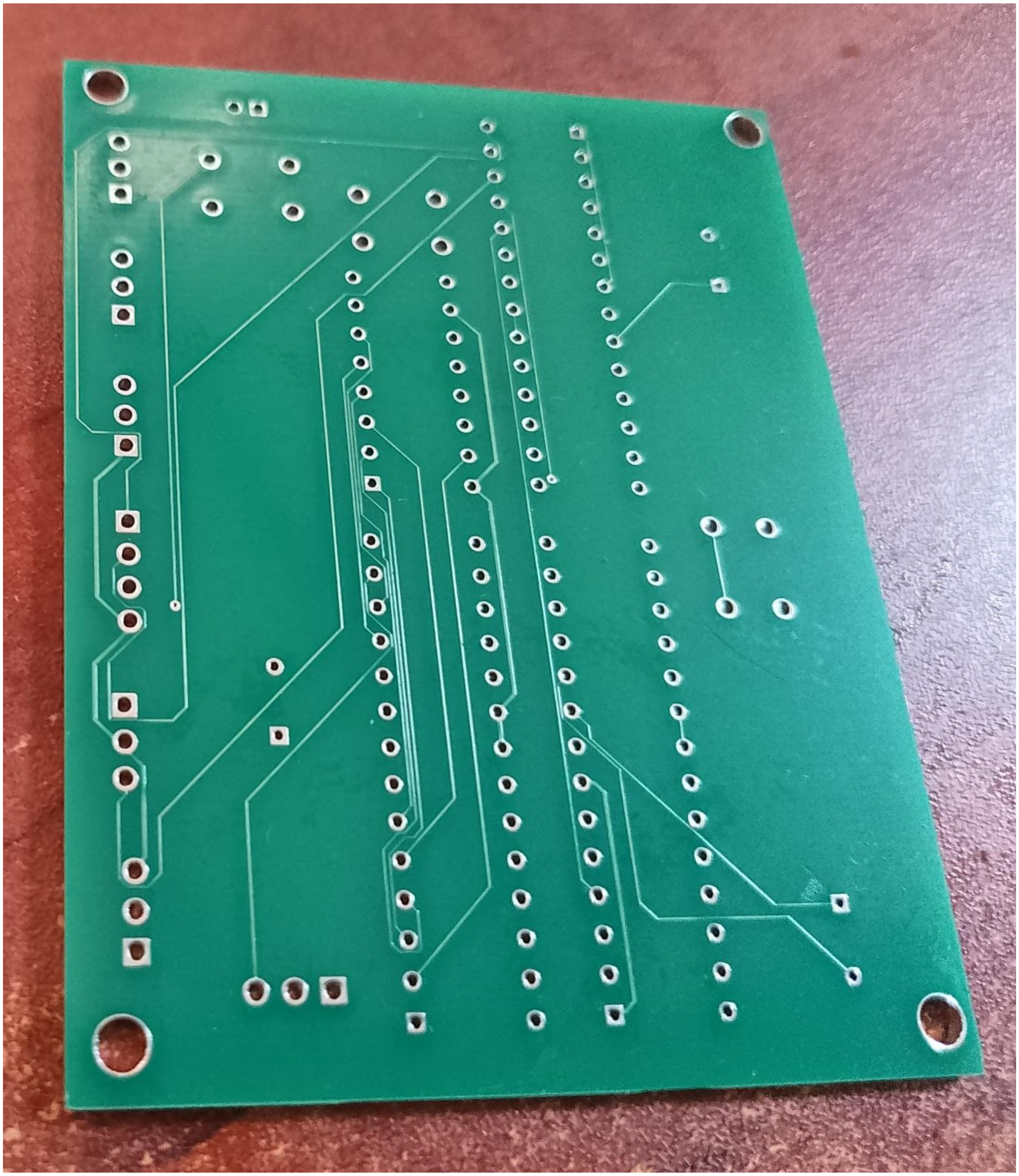


Figure 3

III. Photograph of the bare PCB



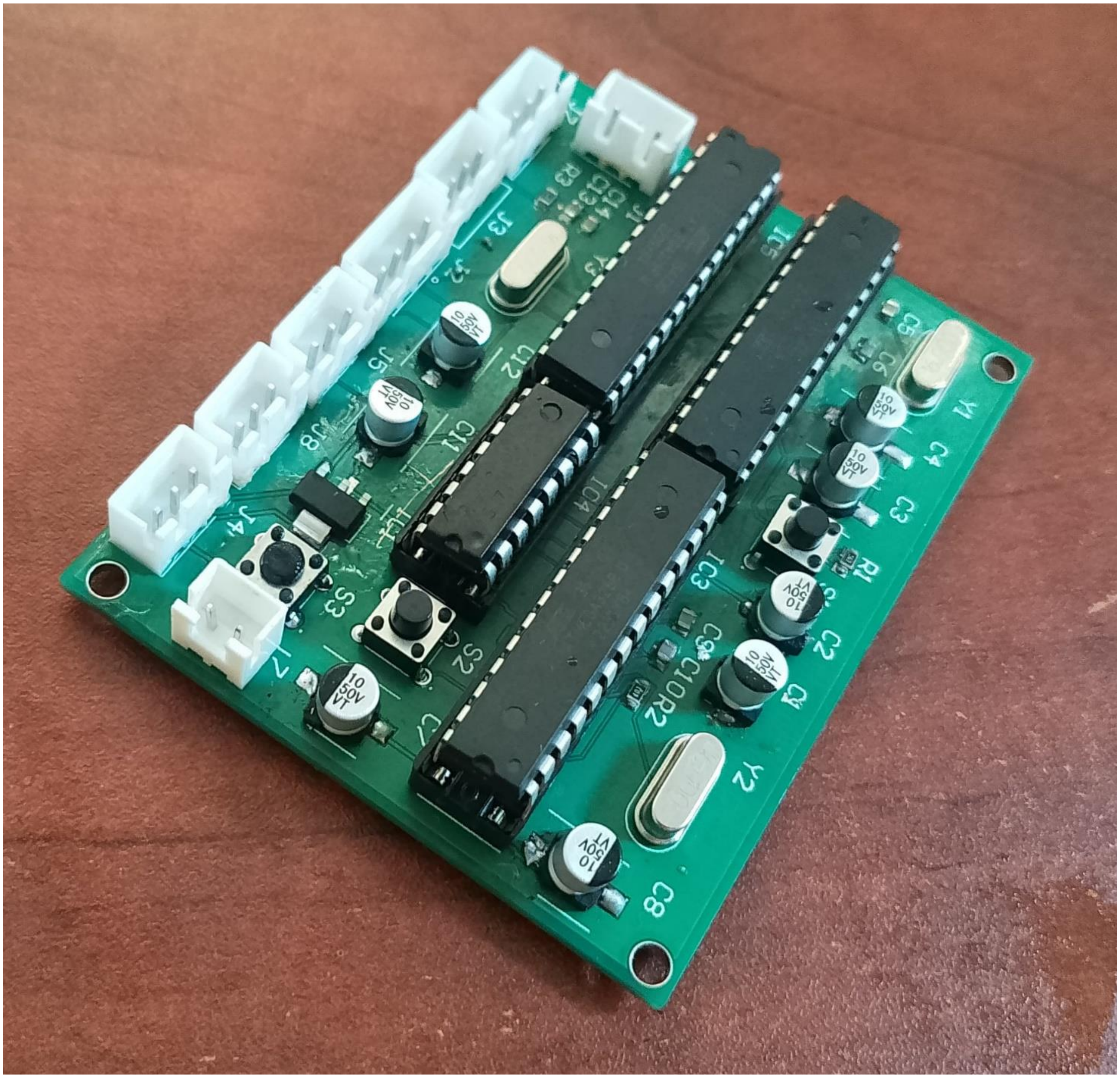
Bare PCB figure 1



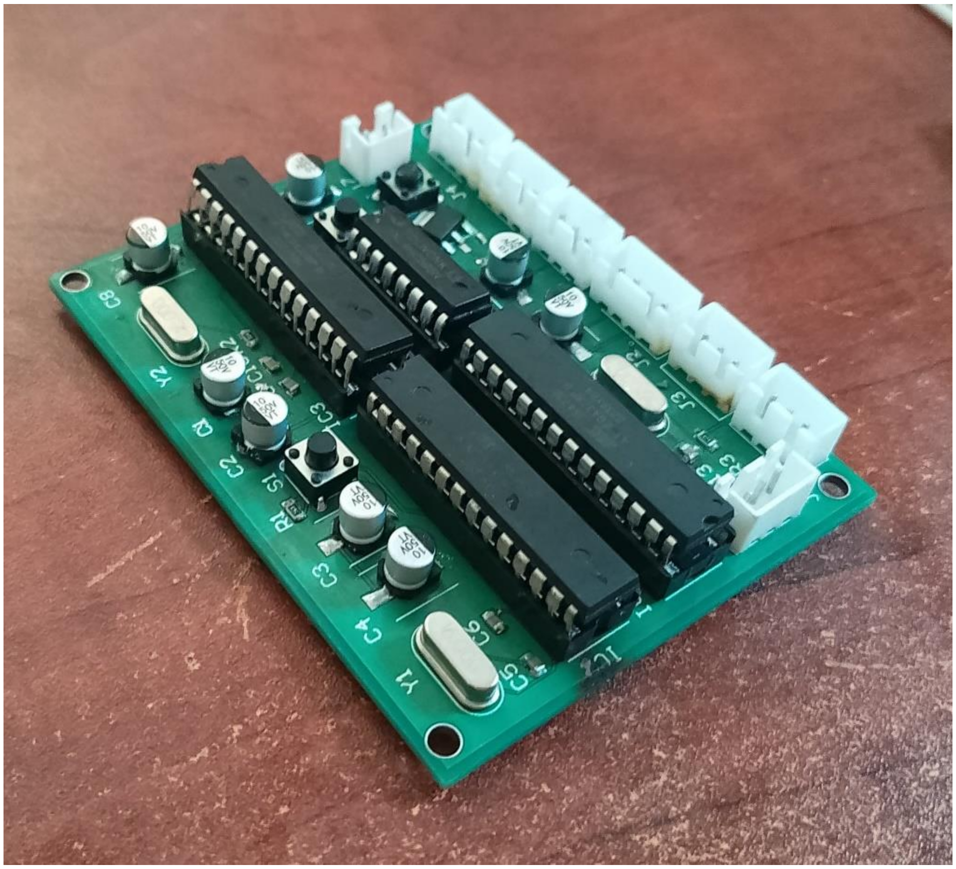
-

Bare PCB figure 2

IV. Photograph of the soldered PCB

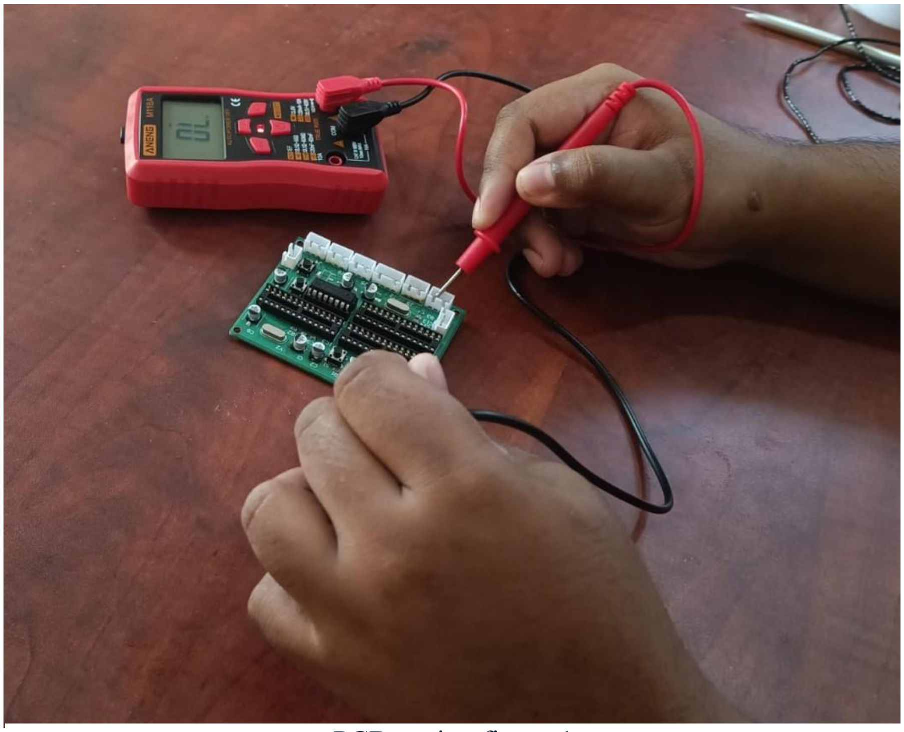


Soldered PCB figure 1

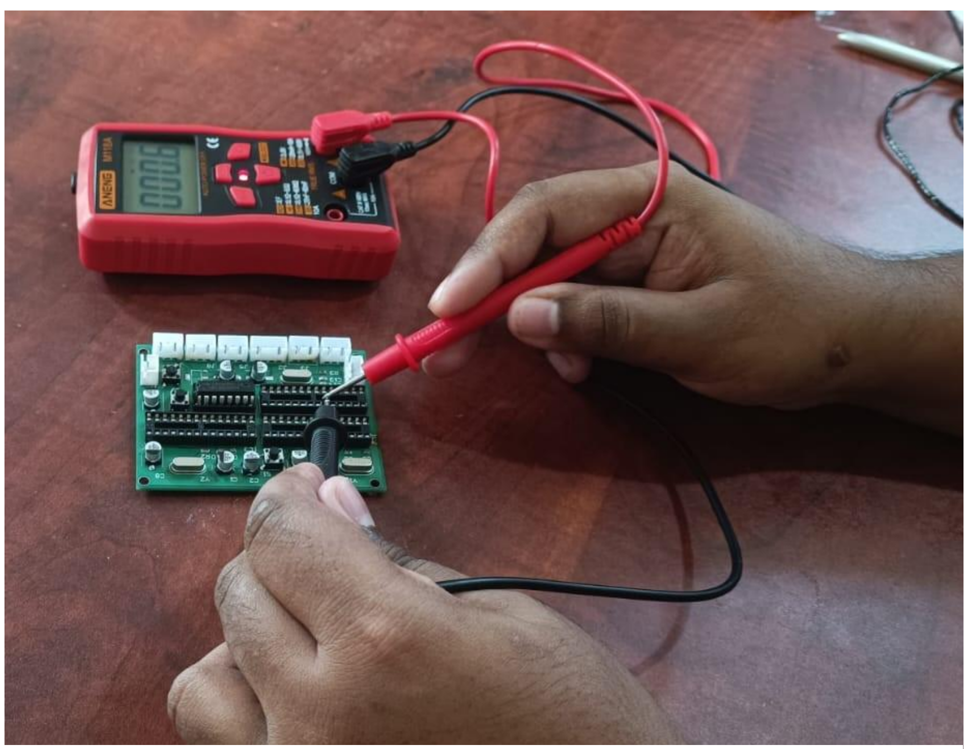


Soldered PCB figure 2

V. Photographs as evidence for PCB testing

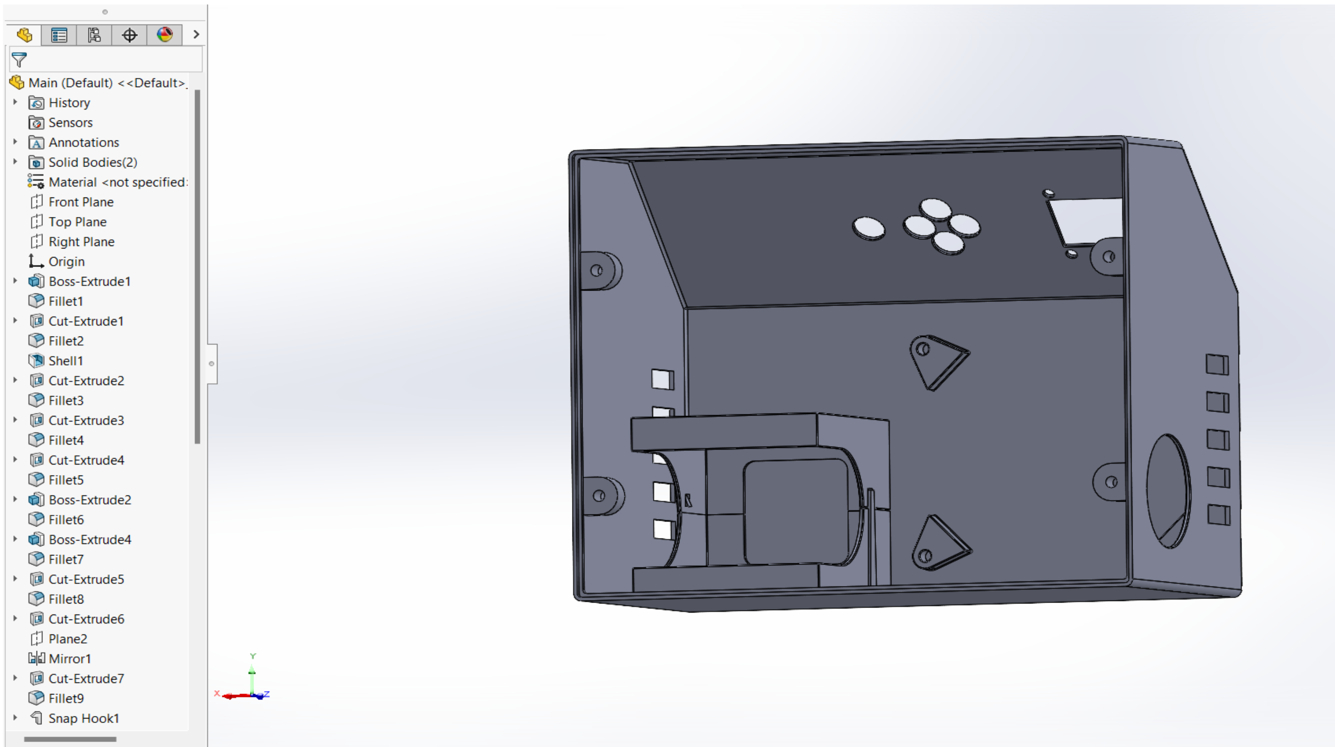


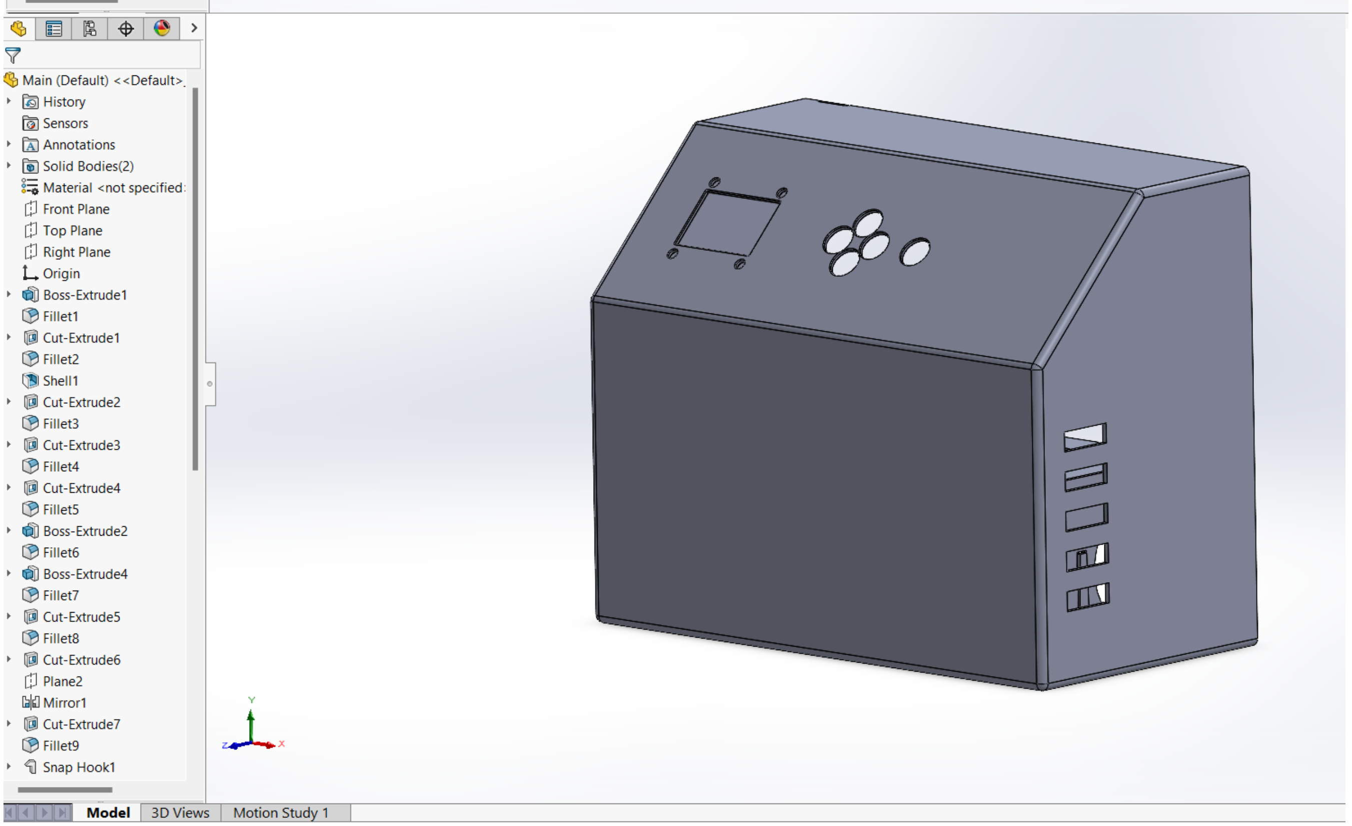
PCB testing figure 1



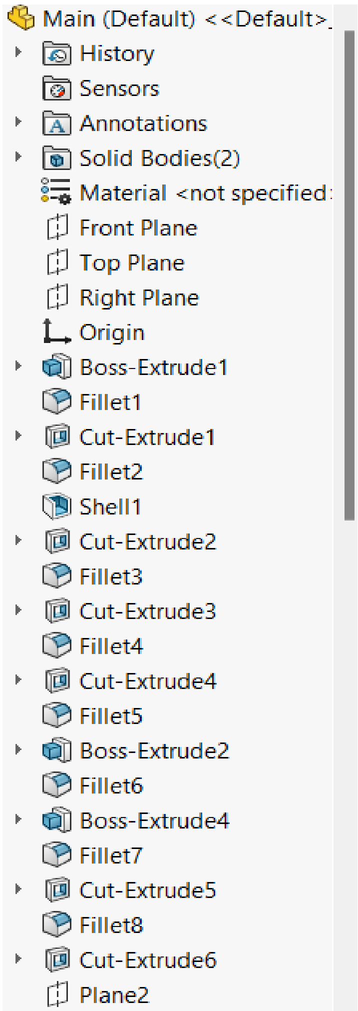
PCB testing figure 2

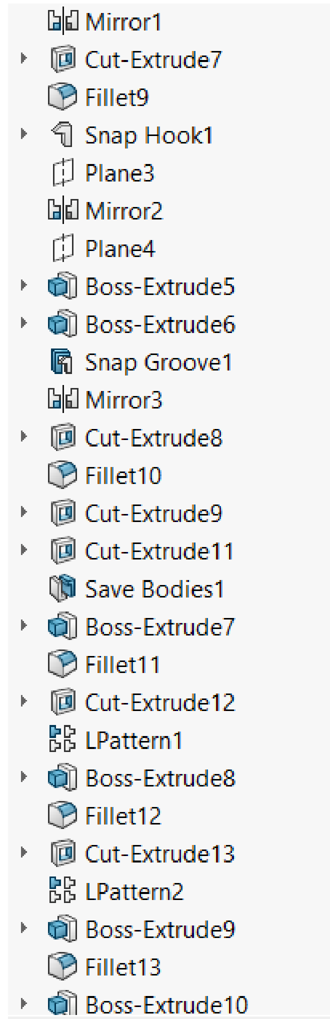
VI. Enclosure Design of the print Controller



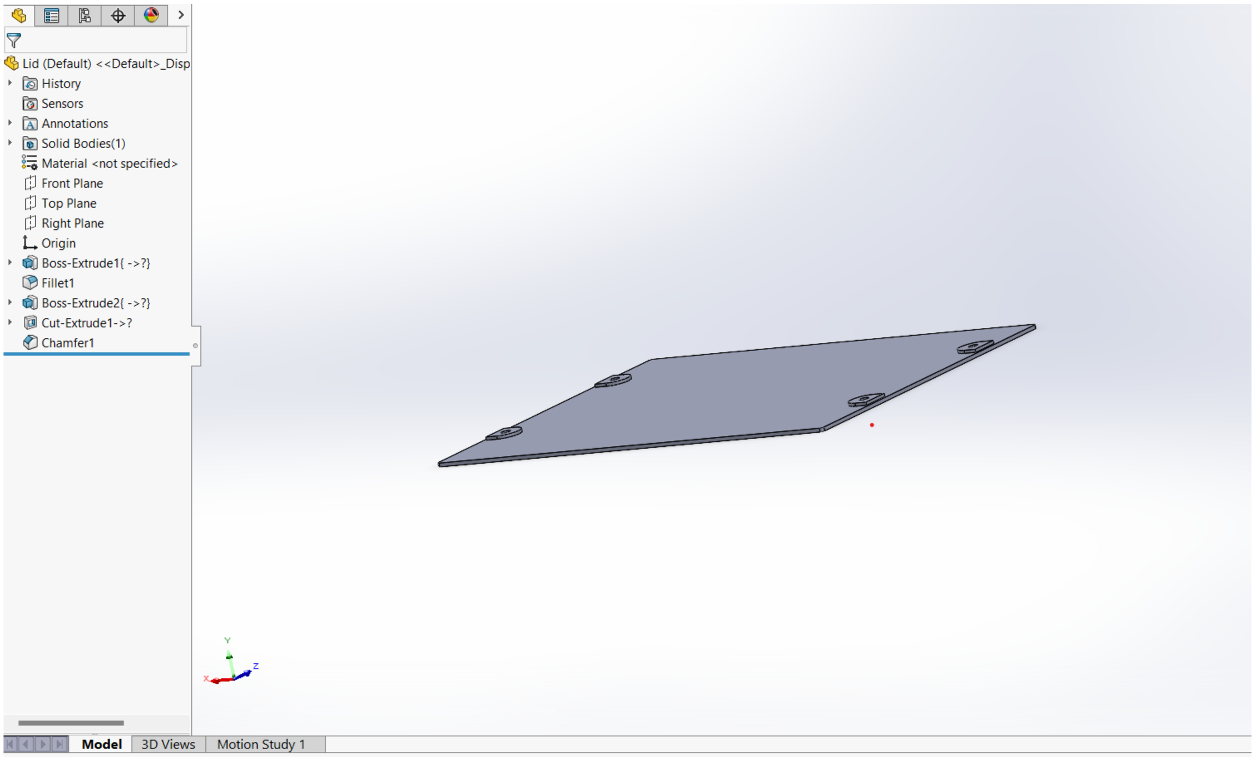


A. Design Tree

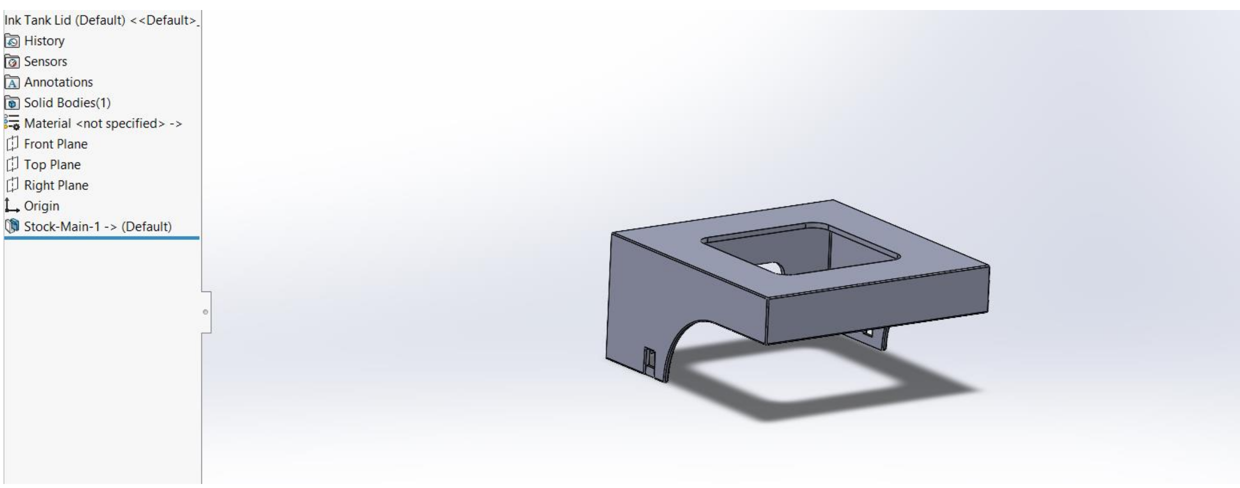




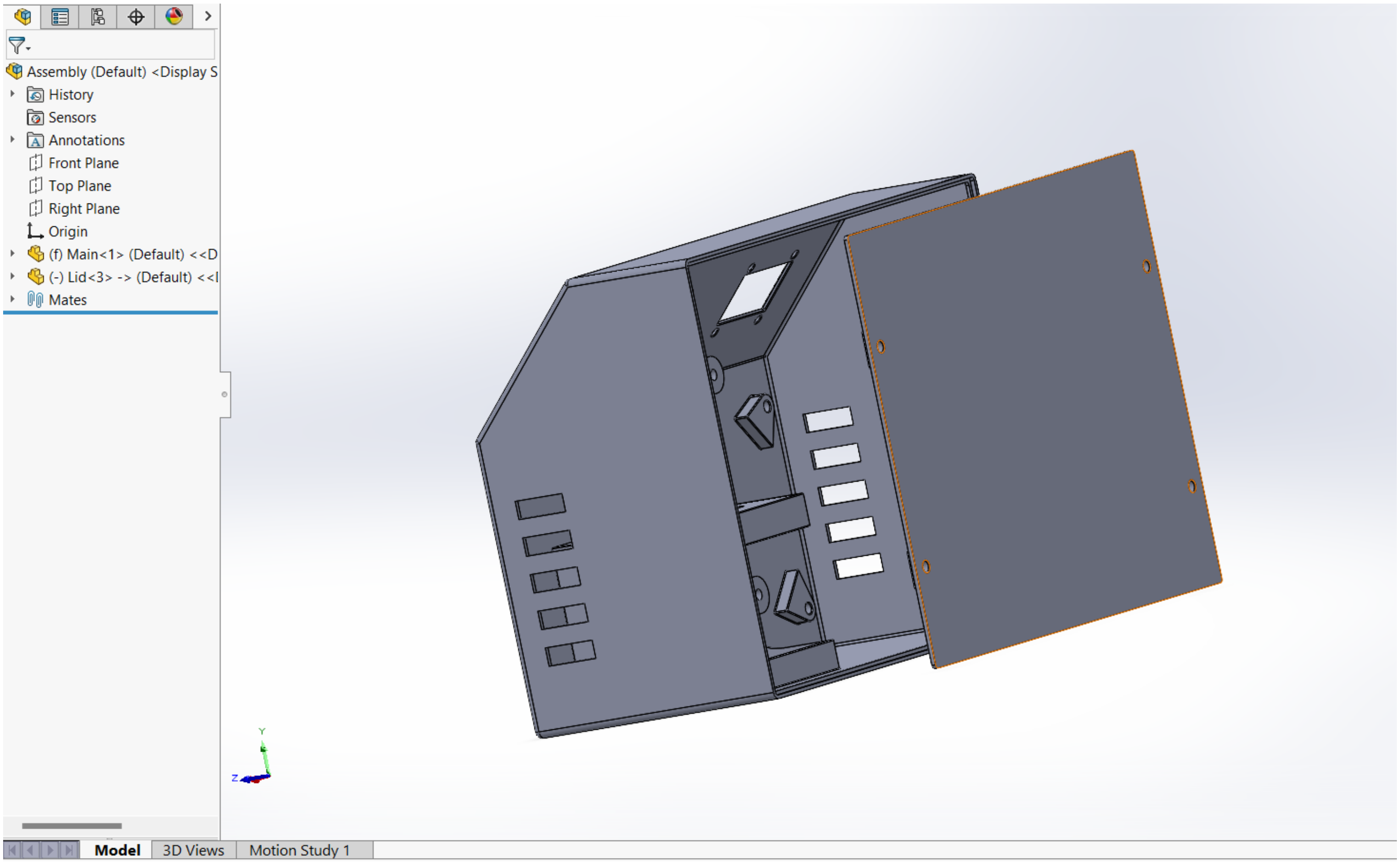
B. Enclosure Lid



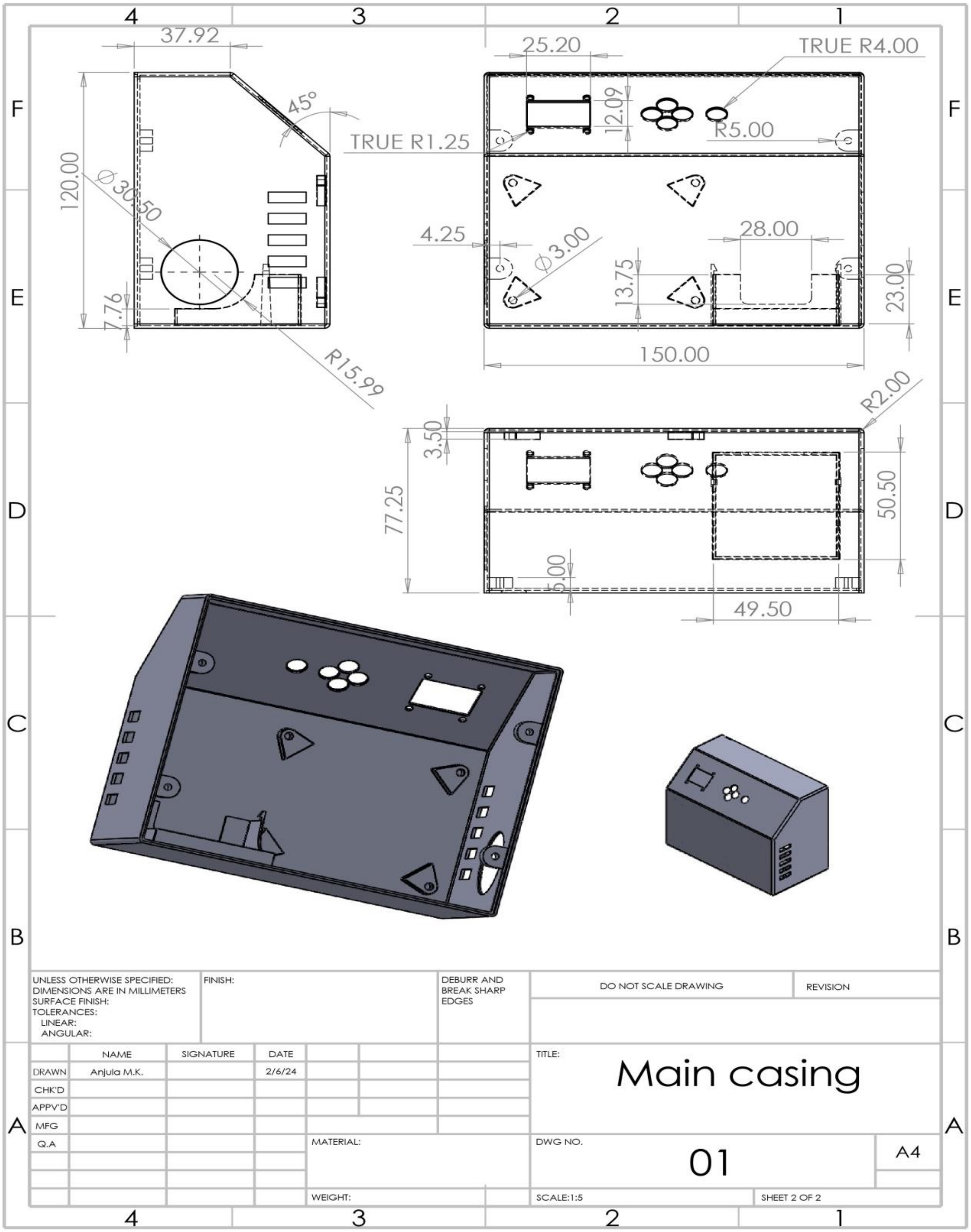
C. Upper Cover of the ink tank inside the main enclosure

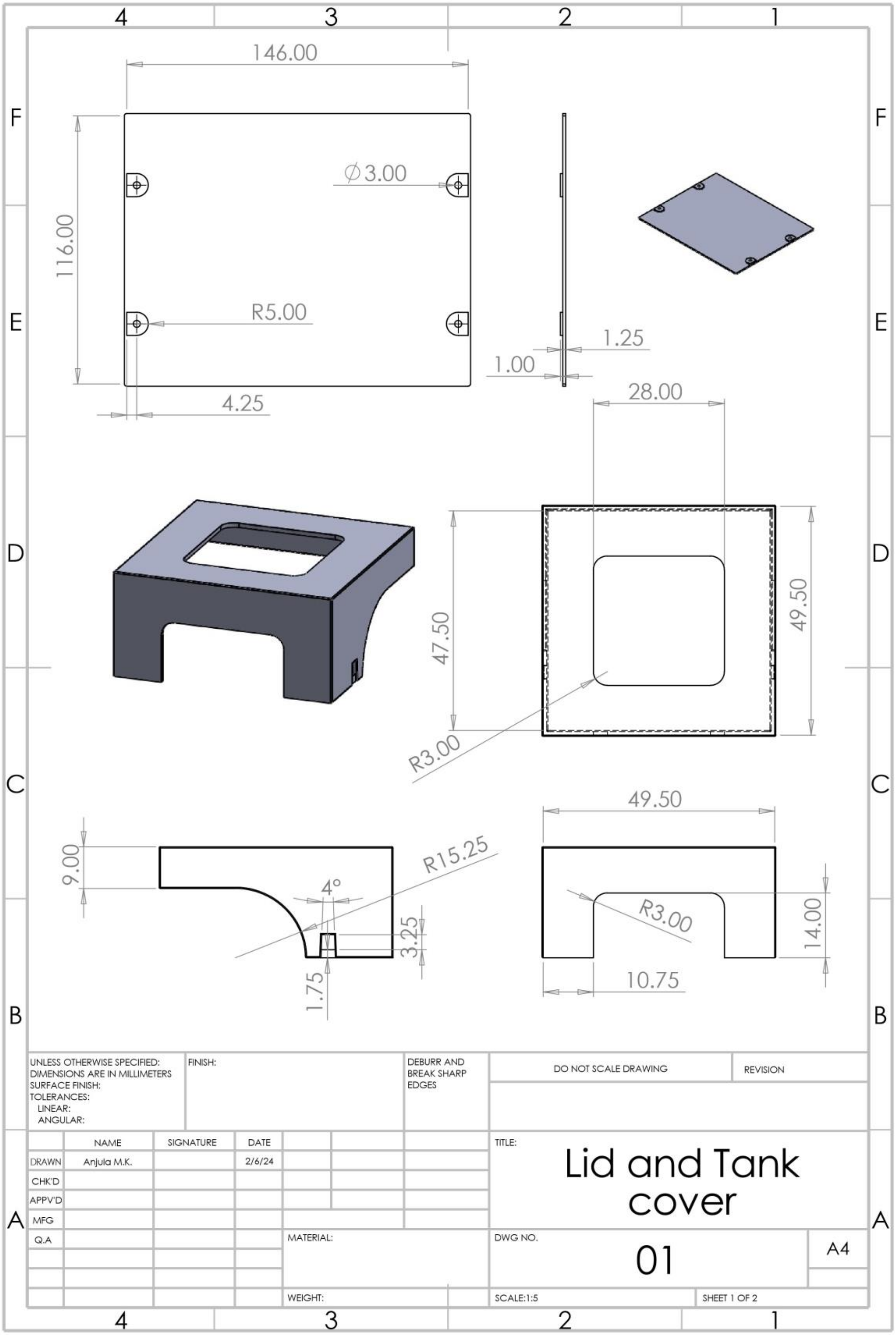


D. Enclosure Assembly



VII. Detailed Design Drawings




146.00
116.00
⌀3.00
R5.00
4.25
1.25
1.00
28.00
47.50
49.50
R3.00
9.00
R15.25
4°
3.25
1.75
49.50
R3.00
14.00
10.75
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
Anjula M.K.
2/6/24
CHK'D
APPV'D
MFG
Q.A
TITLE:
Lid and Tank cover
MATERIAL:
DWG NO.
01
A4
WEIGHT:
SCALE:1:5
SHEET 1 OF 2

VIII. Photographs of the physically built enclosure

Figure 1

Figure 2

Figure 3

Figure 4

IX. Detailed Programming Information

```
#include <avr/io.h>
#include <util/delay.h>
#include <string.h>
#include <avr/pgmspace.h>
#include <U8g2lib.h>

// OLED display size
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

// Define the pins
#define OLED_RESET PB0
#define DATA_PIN PB1
#define CLOCK_PIN PB2
#define LATCH_PIN PB3
#define KEYBOARD_PIN PC0
#define PROXIMITY_PIN PC1
#define PRESSURE_PIN PC2

U8G2_SSD1306_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0);


int New_X = 0;
int Old_X = 0;
int New_Y = 0;
int Old_Y = 0;

// Variable capturing output from Keyboard pin (Values 0 to 1023)
int Key_read = 0;
int Prev_Key_read = 1023;
uint8_t Key_pressed = 0;

// String variable holding the text to transmit
char To_Transmit[100] = "";
int To_Transmit_Length = 0;

// Letter Board
char Letters[3][9] = {
    "ABCDEFGHI",
    "JKLMNOPQR",
    "STUVWXYZ_"
};

// Font8x8 array for ASCII characters
const uint8_t Font8x8[96][8] PROGMEM = {
    {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},   // 0x20 ' '
    {0x18,0x3c,0x3c,0x18,0x18,0x00,0x18,0x00},   // 0x21 '!'
    {0x36,0x36,0x36,0x12,0x00,0x00,0x00,0x00},   // 0x22 '"'
    {0x36,0x36,0x7f,0x36,0x7f,0x36,0x36,0x00},   // 0x23 '#'
    {0x18,0x3e,0x03,0x1e,0x30,0x1f,0x18,0x00},   // 0x24 '$'
    {0x00,0x63,0x33,0x18,0x0c,0x66,0x63,0x00},   // 0x25 '%'
    {0x1c,0x36,0x1c,0x6e,0x3b,0x33,0x6e,0x00},   // 0x26 '&'
    {0x06,0x06,0x03,0x00,0x00,0x00,0x00,0x00},   // 0x27 '''
    {0x18,0x0c,0x06,0x06,0x06,0x0c,0x18,0x00},   // 0x28 '('
    {0x06,0x0c,0x18,0x18,0x18,0x0c,0x06,0x00},   // 0x29 ')'
    {0x00,0x66,0x3c,0xff,0x3c,0x66,0x00,0x00},   // 0x2a '*'
    {0x00,0x18,0x18,0x7e,0x18,0x18,0x00,0x00},   // 0x2b '+'
    {0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x0c},   // 0x2c ','
    {0x00,0x00,0x00,0x7e,0x00,0x00,0x00,0x00},   // 0x2d '-'
    {0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00},   // 0x2e '.'
    {0x60,0x30,0x18,0x0c,0x06,0x03,0x01,0x00},   // 0x2f '/'
    {0x3e,0x63,0x73,0x7b,0x6f,0x67,0x3e,0x00},   // 0x30 '0'
    {0x18,0x1c,0x1e,0x18,0x18,0x18,0x7e,0x00},   // 0x31 '1'
    {0x3e,0x63,0x60,0x3e,0x03,0x63,0x7f,0x00},   // 0x32 '2'
    {0x3e,0x63,0x60,0x3c,0x60,0x63,0x3e,0x00},   // 0x33 '3'
    {0x70,0x78,0x6c,0x66,0x7f,0x60,0xf0,0x00},   // 0x34 '4'
    {0x7f,0x03,0x3f,0x60,0x60,0x63,0x3e,0x00},   // 0x35 '5'
    {0x3e,0x63,0x03,0x3f,0x63,0x63,0x3e,0x00},   // 0x36 '6'
    {0x7f,0x63,0x30,0x18,0x0c,0x0c,0x0c,0x00},   // 0x37 '7'
    {0x3e,0x63,0x63,0x3e,0x63,0x63,0x3e,0x00},   // 0x38 '8'
```

```
    {0x3e,0x63,0x63,0x7e,0x60,0x63,0x3e,0x00},   // 0x39 '9'
    {0x00,0x18,0x18,0x00,0x00,0x18,0x18,0x00},   // 0x3a ':'
    {0x00,0x18,0x18,0x00,0x00,0x18,0x18,0x0c},   // 0x3b ';'
    {0x30,0x18,0x0c,0x06,0x0c,0x18,0x30,0x00},   // 0x3c '<'
    {0x00,0x00,0x7e,0x00,0x00,0x7e,0x00,0x00},   // 0x3d '='
    {0x06,0x0c,0x18,0x30,0x18,0x0c,0x06,0x00},   // 0x3e '>'
    {0x3e,0x63,0x30,0x18,0x0c,0x00,0x0c,0x00},   // 0x3f '?'
    {0x3e,0x63,0x7b,0x7b,0x7b,0x03,0x3e,0x00},   // 0x40 '@'
    {0x18,0x3c,0x66,0x66,0x7e,0x66,0x66,0x00},   // 0x41 'A'
    {0x3f,0x66,0x66,0x3e,0x66,0x66,0x3f,0x00},   // 0x42 'B'
    {0x3c,0x66,0x03,0x03,0x03,0x66,0x3c,0x00},   // 0x43 'C'
    {0x1f,0x36,0x66,0x66,0x66,0x36,0x1f,0x00},   // 0x44 'D'
    {0x7f,0x46,0x16,0x1e,0x16,0x46,0x7f,0x00},   // 0x45 'E'
    {0x7f,0x46,0x16,0x1e,0x16,0x06,0x0f,0x00},   // 0x46 'F'
    {0x3c,0x66,0x03,0x7b,0x63,0x66,0x7c,0x00},   // 0x47 'G'
    {0x66,0x66,0x66,0x7e,0x66,0x66,0x66,0x00},   // 0x48 'H'
    {0x3c,0x18,0x18,0x18,0x18,0x18,0x3c,0x00},   // 0x49 'I'
    {0x78,0x30,0x30,0x30,0x30,0x36,0x1c,0x00},   // 0x4a 'J'
    {0x67,0x66,0x36,0x1e,0x36,0x66,0x67,0x00},   // 0x4b 'K'
    {0x0f,0x06,0x06,0x06,0x46,0x66,0x7f,0x00},   // 0x4c 'L'
    {0x63,0x77,0x7f,0x7f,0x6b,0x63,0x63,0x00},   // 0x4d 'M'
    {0x63,0x67,0x6f,0x7f,0x7b,0x73,0x63,0x00},   // 0x4e 'N'
    {0x3e,0x63,0x63,0x63,0x63,0x63,0x3e,0x00},   // 0x4f 'O'
    {0x3f,0x66,0x66,0x3e,0x06,0x06,0x0f,0x00},   // 0x50 'P'
    {0x3e,0x63,0x63,0x63,0x6b,0x33,0x5e,0x00},   // 0x51 'Q'
    {0x3f,0x66,0x66,0x3e,0x36,0x66,0x67,0x00},   // 0x52 'R'
    {0x3e,0x63,0x07,0x3e,0x70,0x63,0x3e,0x00},   // 0x53 'S'
    {0x7e,0x7e,0x5a,0x18,0x18,0x18,0x3c,0x00},   // 0x54 'T'
    {0x66,0x66,0x66,0x66,0x66,0x66,0x7e,0x00},   // 0x55 'U'
    {0x66,0x66,0x66,0x66,0x66,0x3c,0x18,0x00},   // 0x56 'V'
    {0x63,0x63,0x6b,0x7f,0x7f,0x77,0x63,0x00},   // 0x57 'W'
    {0x63,0x77,0x3e,0x1c,0x3e,0x77,0x63,0x00},   // 0x58 'X'
    {0x66,0x66,0x66,0x3c,0x18,0x18,0x3c,0x00},   // 0x59 'Y'
    {0x7f,0x73,0x31,0x18,0x4c,0x67,0x7f,0x00},   // 0x5a 'Z'
    {0x1e,0x06,0x06,0x06,0x06,0x06,0x1e,0x00},   // 0x5b '['
    {0x03,0x06,0x0c,0x18,0x30,0x60,0x40,0x00},   // 0x5c '\'
    {0x1e,0x18,0x18,0x18,0x18,0x18,0x1e,0x00},   // 0x5d ']'
    {0x08,0x1c,0x36,0x63,0x00,0x00,0x00,0x00},   // 0x5e '^'
    {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff},   // 0x5f '_'
    {0x0c,0x0c,0x18,0x00,0x00,0x00,0x00,0x00},   // 0x60 '`'
    {0x00,0x00,0x3c,0x60,0x7c,0x66,0x7c,0x00},   // 0x61 'a'
    {0x0f,0x06,0x3e,0x66,0x66,0x66,0x3f,0x00},   // 0x62 'b'
    {0x00,0x00,0x3c,0x66,0x06,0x66,0x3c,0x00},   // 0x63 'c'
    {0x78,0x30,0x3e,0x33,0x33,0x33,0x6f,0x00},   // 0x64 'd'
    {0x00,0x00,0x3e,0x63,0x7f,0x03,0x3e,0x00},   // 0x65 'e'
    {0x1c,0x36,0x06,0x0f,0x06,0x06,0x0f,0x00},   // 0x66 'f'
    {0x00,0x00,0x6e,0x33,0x33,0x3e,0x30,0x1f},   // 0x67 'g'
    {0x0f,0x06,0x36,0x6e,0x66,0x66,0x67,0x00},   // 0x68 'h'
    {0x0c,0x00,0x1e,0x0c,0x0c,0x0c,0x3e,0x00},   // 0x69 'i'
    {0x30,0x00,0x30,0x30,0x30,0x30,0x33,0x1e},   // 0x6a 'j'
    {0x0f,0x06,0x66,0x36,0x1e,0x36,0x67,0x00},   // 0x6b 'k'
    {0x1e,0x0c,0x0c,0x0c,0x0c,0x0c,0x3e,0x00},   // 0x6c 'l'
    {0x00,0x00,0x33,0x7f,0x7f,0x6b,0x63,0x00},   // 0x6d 'm'
    {0x00,0x00,0x1f,0x33,0x33,0x33,0x33,0x00},   // 0x6e 'n'
    {0x00,0x00,0x3e,0x63,0x63,0x63,0x3e,0x00},   // 0x6f 'o'
    {0x00,0x00,0x3b,0x66,0x66,0x3e,0x06,0x0f},   // 0x70 'p'
    {0x00,0x00,0x6e,0x33,0x33,0x3e,0x30,0x78},   // 0x71 'q'
    {0x00,0x00,0x3b,0x36,0x36,0x0e,0x1f,0x00},   // 0x72 'r'
    {0x00,0x00,0x3e,0x03,0x3e,0x60,0x3f,0x00},   // 0x73 's'
    {0x08,0x0c,0x3f,0x0c,0x0c,0x2c,0x18,0x00},   // 0x74 't'
    {0x00,0x00,0x33,0x33,0x33,0x33,0x6e,0x00},   // 0x75 'u'
    {0x00,0x00,0x33,0x33,0x33,0x1e,0x0c,0x00},   // 0x76 'v'
    {0x00,0x00,0x63,0x6b,0x7f,0x7f,0x36,0x00},   // 0x77 'w'
    {0x00,0x00,0x63,0x36,0x1c,0x36,0x63,0x00},   // 0x78 'x'
    {0x00,0x00,0x33,0x33,0x33,0x3e,0x30,0x1f},   // 0x79 'y'
    {0x00,0x00,0x3f,0x19,0x0c,0x26,0x3f,0x00},   // 0x7a 'z'
    {0x38,0x0c,0x0c,0x07,0x0c,0x0c,0x38,0x00},   // 0x7b '{'
    {0x18,0x18,0x18,0x00,0x18,0x18,0x18,0x00},   // 0x7c '|'
    {0x07,0x0c,0x0c,0x38,0x0c,0x0c,0x07,0x00},   // 0x7d '}'
    {0x6e,0x3b,0x00,0x00,0x00,0x00,0x00,0x00},   // 0x7e '~'
```

```
    {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}    // 0x7f 'DEL'
};
void init_pins() {
    // Initialize IO pins
    DDRB |= (1 << DATA_PIN) | (1 << CLOCK_PIN) | (1 << LATCH_PIN);  // Set shift register pins as output
    DDRB &= ~(1 << KEYBOARD_PIN);  // Set keyboard pin as input
    DDRB &= ~(1 << PROXIMITY_PIN); // Set proximity sensor pin as input
    DDRB &= ~(1 << PRESSURE_PIN);  // Set pressure sensor pin as input
}
void setup() {
    // Initialize
    init_pins();
    init_display();
    init_keyboard();
    init_sensors();
}

void loop() {
        for (int i = 0; input_text[i] != '\0'; ++i) {
            char ascii_char = input_text[i];
            uint8_t bitmap[8];
            convert_ascii_to_bitmap(ascii_char, bitmap);
            for (int j = 0; j < 8; ++j) {
                send_to_shift_register(bitmap[j]);
            }
        }
       displayToShiftRegister();
       delay_ms(1000);
}



int main() {
    init_display();
    readSensors();
    char* input_text = get_user_input();


        process_text(input_text);

        // Display the processed text on the OLED display
        refresh_display(input_text);


        uint8_t sensor_data = (uint8_t)(ADC / 4);
    if (Key_read != Prev_Key_read) {
        if (Key_read) {
            Key_pressed = 1;
            }
        Prev_Key_read = Key_read;
    }
    processText();
    while(1){
      loop();
    }
    return 0;
}


void refreshDisplay(void) {
    u8g2_ClearBuffer(&u8g2);
    u8g2_DrawStr(&u8g2, 0, 10, "Select Letter:");
    for (int y = 0; y < 3; y++) {
      for (int x = 0; x < 9; x++) {
            if (New_X == x && New_Y == y) {
                u8g2_DrawBox(&u8g2, x * 8 + 2, y * 8 + 12, 8, 8);
                u8g2_SetDrawColor(&u8g2, 0);
                u8g2_DrawGlyph(&u8g2, x * 8 + 2, y * 8 + 20, Letters[y][x]);
                u8g2_SetDrawColor(&u8g2, 1);
            } else {
```

```
                u8g2_DrawGlyph(&u8g2, x * 8 + 2, y * 8 + 20, Letters[y][x]);
            }
        }
    }
    u8g2_SendBuffer(&u8g2);
}

void shiftOut(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t val) {
    for (uint8_t i = 0; i < 8; i++) {
        if (bitOrder == LSBFIRST) {
            if (val & (1 << i))
                PORTD |= (1 << dataPin);
            else
                PORTD &= ~(1 << dataPin);
        } else {
            if (val & (1 << (7 - i)))
                PORTD |= (1 << dataPin);
            else
                PORTD &= ~(1 << dataPin);
        }
        PORTD |= (1 << clockPin);
        PORTD &= ~(1 << clockPin);
    }
}

void displayToShiftRegister(void) {
    for (int i = 0; i < To_Transmit_Length; i++) {
        char c = To_Transmit[i];
        uint8_t asciiIndex = c - 32;
        for (int j = 0; j < 8; j++) {
            shiftOut(DATA_PIN, CLOCK_PIN, MSBFIRST, pgm_read_byte(&(Font8x8[asciiIndex][j])));
        }
    }
}

void processText(void) {
    if (Key_pressed) {
        char selectedLetter = Letters[New_Y][New_X];
        if (selectedLetter != '_') {
            To_Transmit[To_Transmit_Length++] = selectedLetter;
        } else if (To_Transmit_Length > 0) {
            To_Transmit[--To_Transmit_Length] = '\0';
        }
        Key_pressed = 0;
    }
}

void readSensors(void) {
    uint16_t proximityValue = ADC_read(PROXIMITY_SENSOR_PIN);
    uint16_t pressureValue = ADC_read(PRESSURE_SENSOR_PIN);
    if (proximityValue > 512) {
        Key_read = 1;
    } else if (pressureValue > 512) {
        Key_read = 1;
    } else {
        Key_read = 0;
    }
}

uint16_t ADC_read(uint8_t ch) {
    ch &= 0b00000111;
    ADMUX = (ADMUX & 0xF8) | ch;
    ADCSRA |= (1 << ADSC);
    while (ADCSRA & (1 << ADSC));
    return (ADC);
}



void init_display() {
```

```
  u8g2_Setup_ssd1306_128x64_noname_f(&u8g2, U8G2_R0, u8x8_byte_avr_spi, u8x8_avr_delay);
  u8x8_SetPin(u8g2_GetU8x8(&u8g2), U8X8_PIN_DC, PD2);
  u8x8_SetPin(u8g2_GetU8x8(&u8g2), U8X8_PIN_RESET, OLED_RESET);
  u8x8_SetPin(u8g2_GetU8x8(&u8g2), U8X8_PIN_CS, PD3);
  u8g2_InitDisplay(&u8g2);
  u8g2_SetPowerSave(&u8g2, 0);

}



void init_sensors() {

    ADMUX = (1 << REFS0);
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1);
}

void refresh_display(const char* text) {
    // Clear display
    u8g2_ClearBuffer(&u8g2);

    // Set font and position
    u8g2_SetFont(&u8g2, u8g2_font_8x8_tf);
    u8g2_SetCursor(&u8g2, 0, 10);

    // Display text on OLED
    u8g2_DrawStr(&u8g2, 0, 16, text);

    // Send buffer to OLED
    u8g2_SendBuffer(&u8g2);
}

char* get_user_input() {
    static char input_text[32];
    strcpy(input_text,To_Transmit);

    return input_text;
}

void convert_ascii_to_bitmap(char ascii_char, uint8_t* bitmap) {
    // Convert ASCII character to bitmap using Font8x8 array
    if (ascii_char < 32 || ascii_char > 127) {
        ascii_char = 32;
    }
    memcpy(bitmap, Font8x8[ascii_char - 32], 8);
}

void send_to_shift_register(uint8_t data) {
    PORTB |= (1 << LATCH_PIN);   // Set latch pin high
    for (int i = 7; i >= 0; --i) {
        if (data & (1 << i)) {
            PORTB |= (1 << DATA_PIN);  // Set data pin high
        } else {
            PORTB &= ~(1 << DATA_PIN); // Set data pin low
        }
        PORTB |= (1 << CLOCK_PIN);  // Clock pin high
        PORTB &= ~(1 << CLOCK_PIN); // Clock pin low
    }
    PORTB &= ~(1 << LATCH_PIN);   // Set latch pin low
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//pressure controller

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

#define F_CPU 16000000UL

```

```
#define PRESSURE_SENSOR_PIN 5
#define PWM_PIN 4
#define STATUS_PIN 2

#define ADC_MAX_VALUE 1023
#define V_REF 5.0
#define IDEAL_VOLTAGE 2.2
#define MIN_VOLTAGE 1.2
#define MAX_VOLTAGE 3.2

class ADC {
public:
    static void setup() {
        // Select Vref=AVcc
        ADMUX = (1 << REFS0);
        // Set ADC prescaler to 128 for 125kHz ADC clock
        ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
    }

    static uint16_t read(uint8_t channel) {
        // Select ADC channel
        ADMUX = (ADMUX & 0xF0) | (channel & 0x0F);
        // Start conversion
        ADCSRA |= (1 << ADSC);
        // Wait for conversion to complete
        while (ADCSRA & (1 << ADSC));
        return ADC;
    }
};

class PWM {
public:
    static void setup() {
        // Set PWM_PIN as output
        DDRD |= (1 << DDD4);
        // Set Timer1 to Fast PWM mode, non-inverted
        TCCR1A = (1 << COM1B1) | (1 << WGM11);
        TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS11); // Prescaler 8
        ICR1 = 39999; // Set TOP value for 20ms period (50Hz)
    }

    static void setDutyCycle(uint8_t dutyCycle) {
        OCR1B = (ICR1 * dutyCycle) / 100;
    }
};

class StatusPin {
public:
    static void setup() {
        // Set STATUS_PIN as output
        DDRD |= (1 << DDD2);
    }

    static void set(uint8_t status) {
        if (status) {
            PORTD |= (1 << PORTD2);
        } else {
            PORTD &= ~(1 << PORTD2);
        }
    }
};

class PressureControl {
public:
    static void run() {
        ADC::setup();
        PWM::setup();
        StatusPin::setup();

        while (true) {
```

```
            // Read the analog value from the pressure sensor
            uint16_t adcValue = ADC::read(PRESSURE_SENSOR_PIN);
            // Convert ADC value to voltage
            float voltage = (adcValue * V_REF) / ADC_MAX_VALUE;
            // Calculate PWM duty cycle
            float dutyCycle = 50.0 + (IDEAL_VOLTAGE - voltage) * 50.0;

            if (voltage < MIN_VOLTAGE || voltage > MAX_VOLTAGE) {
                StatusPin::set(0);
            } else {
                StatusPin::set(1);
                if (voltage < IDEAL_VOLTAGE) {
                    dutyCycle += (IDEAL_VOLTAGE - voltage) * 50.0 / (IDEAL_VOLTAGE - MIN_VOLTAGE);
                } else if (voltage > IDEAL_VOLTAGE) {
                    dutyCycle -= (voltage - IDEAL_VOLTAGE) * 50.0 / (MAX_VOLTAGE - IDEAL_VOLTAGE);
                }
            }

            if (dutyCycle > 100) dutyCycle = 100;
            if (dutyCycle < 0) dutyCycle = 0;

            PWM::setDutyCycle(static_cast<uint8_t>(dutyCycle));
            _delay_ms(100);
        }
    }
};

int main() {
    PressureControl::run();
    return 0;
}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////PWM generator

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

#define F_CPU 16000000UL

class PWMController {
public:
    PWMController() {
        // Set digital pins 0-7 as inputs
        DDRD &= ~0xFF;  // Clear lower 8 bits of DDRD to set PORTD0 to PORTD7 as inputs

        // Set digital pin 8 as output (for PWM signal)
        DDRB |= (1 << DDB0);  // Set PORTB0 as output (digital pin 8)

        // Set up Timer0 for PWM
        TCCR0A |= (1 << COM0A1) | (1 << WGM00) | (1 << WGM01);  // Fast PWM mode, non-inverted
        TCCR0B |= (1 << CS01);  // Prescaler set to 8
    }

    void setPWMDutyCycle(uint8_t dutyCycle) {
        OCR0A = (dutyCycle * 255) / 100;  // Set PWM duty cycle
    }

    uint8_t checkPins() {
        for (uint8_t i = 0; i < 8; i++) {
            if (PIND & (1 << i)) {
                return (i + 1) * 10;  // Return duty cycle based on pin level (10%, 20%, ..., 80%)
            }
        }
        return 0;  // If no pins are high, return 0% duty cycle
    }

    void run() {
        while (true) {
            uint8_t dutyCycle = checkPins();
```

```
            setPWMDutyCycle(dutyCycle);
            _delay_ms(100);  // Adjust this value to control the cycle speed
        }
    }
};

int main() {
    PWMController pwmController;
    pwmController.run();
    return 0;
}
```

X. Daily Log Entries

Project Title: Automatic Batch Code Printing Machine Development

01/02/2024 - Project Selection (Part 1)

- Organized a project kickoff meeting.
- Conducted market research to identify potential improvements within the industry, generating several project ideas and proposals.

04/02/2024 - Project Selection (Part 2)

- Selected the best project idea to support local industries, focusing on an automatic batch code printing machine.
- Conducted preliminary research on current batch code printing machines, noting key features and functionalities.

10/02/2024 - Project Evaluation

- Conducted a self-evaluation of the selected project.

16/02/2024 - Self Review

- Conducted market research to identify stakeholders, users, and competitors.

28/02/2024 - Generating Preliminary Designs

- Developed various stimulated ideas for different printing techniques to achieve the required output.
- Created three different conceptual designs based on these ideas.

04/03/2024 - Evaluation of Conceptual Designs

- Selected the best conceptual design for further progress based on evaluation criteria.

12/03/2024 - Designing of the Circuit (Part 1)

- Began designing the circuit layout, focusing on organizing key components such as input devices (keyboard, display), power supply, pressure sensors, and proximity sensors.
- Sketched initial circuit layouts on paper, outlining potential pathways for power and signal flow.
- Explored different methods of obtaining the control signal required to generate the electric field.

20/03/2024 - Designing of the Circuit (Part 2)

- Selected the best circuit components suited to the initial functions of the batch code printer.

21/03/2024 - Designing of the Circuit (Part 3)

- Created preliminary schematics of the circuit layout to ensure proper connections between all components.
- Further refined the circuit design, integrating the microcontroller and communication interfaces.

25/03/2024 - Program Implementation (Part 1)

- Designed an algorithm to map ASCII characters of given text to their respective bitmap representation in a dot matrix.
- Developed an algorithm to shift the bitmap representation of each character column by column to achieve the scrolling effect.
- Developed C++ code for the designed algorithm.

28/03/2024 - Program Implementation (Part 2)

- Designed an algorithm to establish the connection between sensors and handle user inputs, and developed the corresponding C++ code.
- Finalized the implemented code.

07/04/2024 - Schematic Design (Part 1)

- Began developing digital schematics, concentrating on logical arrangement for clarity and ease of understanding.

11/04/2024 - Schematic Design (Part 2)

- Continued developing and refining the schematic design to ensure all essential components and connections are accurately depicted.
- Held a design review with the team to ensure schematic consistency and completeness.
- Incorporated necessary adjustments based on feedback from the supervisor.

17/04/2024 - PCB Layout (Part 1)

- Transitioned from schematic design to PCB layout, beginning with component placement on the board.
- Emphasized optimal positioning to maintain signal integrity and manage power effectively.
- Considered size limitations and arrangement of crucial interfaces.

19/04/2024 - PCB Layout (Part 2)

- Continued routing connections between components, ensuring all signal paths are optimized for minimal interference and maximum signal integrity.
- Used design rules and guidelines to meet industry standards and manufacturing requirements.
- Implemented design techniques such as impedance matching for high-speed signal paths to minimize reflections and signal degradation.
- Addressed potential noise sources, such as switching regulators or high-speed digital signals, and implemented strategies to mitigate their effects on sensitive analog and digital circuits.
- Verified that all trace widths and clearances were within specifications to handle both power and signal lines effectively.

21/04/2024 - PCB Layout (Part 3)

- Double-checked the layout for any potential design rule violations and corrected them as needed.

- Ensured all components are correctly placed and oriented according to the design specifications.
- Conducted an exhaustive design review, meticulously verifying the PCB layout against the schematic and project requirements.
- Implemented optimizations to enhance manufacturability and facilitate ease of assembly.

23/04/2024 - Enclosure Design (Part 1)

- Discussed the enclosure requirements, covering aspects like dimensions, material selection, and functionality requirements for the print head.
- Developed initial design sketches for the enclosure, considering optimal placement options for the pressure tank, display, pressure valves, and PCB designs.
- Explored various enclosure materials, evaluating factors such as cost-effectiveness, durability, and suitability for environmental conditions.

25/04/2024 - Enclosure Design (Part 2)

- Implemented design modifications to incorporate connectors, vents, and mounting features.
- Integrated cable management solutions to ensure tidy organization within the enclosure.

28/04/2024 - Enclosure Design (Part 3)

- Conducted a design review meeting to evaluate the enclosure's compatibility with the PCB and overall system.
- Finalized the enclosure design.

29/04/2024 - PCB Production & Component Ordering

- Submitted the finalized PCB design files to a manufacturer for production, ensuring adherence to manufacturing guidelines and specifications.
- Ordered necessary components, including resistors, capacitors, ICs, connectors, and other required sensors.
- Reviewed and updated the Bill of Materials (BOM) to reflect any changes or substitutions made during the component ordering process.

07/05/2024 - Component Arrival & Preparation

- Received the manufactured PCBs and all ordered components.
- Conducted a thorough inspection of the PCBs to ensure they meet quality standards and comply with design specifications.
- Organized and categorized the components to facilitate smooth and efficient assembly processes.

10/05/2024 - PCB Assembly & Soldering (Part 1)

- Started assembling components onto the PCB following design specifications.
- Carefully placed and soldered each component, starting with smaller ones and progressing to larger ones.
- Conducted intermediate visual inspections to check for any soldering defects or misalignments.
- Checked solder joints for continuity to ensure proper electrical connections.
- Completed assembly and soldering of the PCB.
- Cleaned assembled PCBs to remove any remaining flux residue and inspected them for any remaining issues.

12/05/2024 - Project Final Documentation

- Created comprehensive documentation covering the entire project from inception to completion.

XI. References

[1] alldatasheet.com, “AMS1117-5.0 PDF,” *pdf1.alldatasheet.com*.

https://pdf1.alldatasheet.com/datasheet-pdf/view/205692/ADMOS/AMS1117-5.0.html
 (accessed Jul. 05, 2024).

[2]
“XH CONNECTOR.” Available: https://www.jst-mfg.com/product/pdf/eng/eXH.pdf

[3]“ATmega328P 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash DATASHEET.” Available:

https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

[4]“B3F Tactile Switch Through-hole-mounting Switches in a Wide Range of Models: 6 × 6 mm, 12 × 12 mm, Side-operated Models, Gold-plated Contacts, and Radial Tape.” Available: https://www.mouser.com/datasheet/2/307/en-b3f-13826.pdf

[5]“Hidden Creek Ln Spicewood TX 78669 5101.” Accessed: Jul. 05, 2024. [Online]. Available: https://www.mouser.com/datasheet/2/3/ABL-1774766.pdf

[6]“General Specifications GENERAL DESCRIPTION Automotive MLCC with FLEXITERM ®.” Accessed: Jul. 05, 2024. [Online]. Available: https://www.mouser.com/datasheet/2/40/AutoMLCCKAF-3218225.pdf

XII. Appendix: Initial Code by Arduino

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>


#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64


  int Keyboard=A7;


  int New_X=0;
  int Old_X=0;
  int New_Y=0;
  int Old_Y=0;


  int Key_read=0;
  int Prev_Key_read=1023;
  boolean Key_pressed=false;

// String variable holding the text to transmit
char To_Transmit="";

// Length of the text to transmit
int To_Transmit_Length=0;



  #define OLED_RESET     4
  Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);


// Used for displaying Leter board
char Letters[3][9]={"ABCDEFGHI",
                    "JKLMNOPQR",
                    "STUVWXYZ_"};
```

```
char msg[] ="";
int scrollspeed = 35;

int x;
int y;

// Rows (Vertical control)
int clockPin2 = 2; //  pin connected to Clock Pin 11 of 74HC595
int latchPin2 = 7; //  pin connected to Latch Pin 12 of 74HC595
int dataPin2 = 3;
int psensor= 24;
int proxy = 25;

// FONT DEFINITION
byte alphabets[][5] = {
 {0,0,0,0,0},                    // space      ASCII 32
 {0,0,253,0,0},                  // !           ASCII 33
 {0,96,0,96,0},                   // "           ASCII 34
 {20,127,20,127,20},             // #           ASCII 35
 {36,42,127,42,18},                 // $           ASCII 36
 {17,2,4,8,17},                     // %           ASCII 37
 {54,73,85,34,5},                   // &           ASCII 38
 {0,0,104,112,0},                   // '           ASCII 39
 {28,34,65},                        // (           ASCII 40
 {65,34,28},                        // )           ASCII 41
 {20,8,62,8,20},                    // *           ASCII 42
 {8,8,62,8,8},                      // +           ASCII 43
 {0,0,5,6,0},                       // ,           ASCII 44
 {8,8,8,8,8},                       // -           ASCII 45
 {0,0,1,0,0},                       // .           ASCII 46
 {1,2,4,8,16},                      // /           ASCII 47
 {62,69,73,81,62},                  // 0           ASCII 48
 {0,33,127,1,0},                    // 1           ASCII 49
 {33,67,69,73,49},                  // 2           ASCII 50
 {66,65,81,105,70},                 // 3           ASCII 51
 {12,20,36,127,4},                  // 4           ASCII 52
 {113,81,81,81,78},                 // 5           ASCII 53
 {30,41,73,73,6},                   // 6           ASCII 54
 {64,64,79,80,96},                  // 7           ASCII 55
 {54,73,73,73,54},                  // 8           ASCII 56
 {48,73,73,74,60},                  // 9           ASCII 57
 {0,0,54,54,0},                     // :           ASCII 58
 {0,0,53,54,0},                     // ;           ASCII 59
 {0,8,20,34,65},                    // <           ASCII 60
 {20,20,20,20,20},                  // =           ASCII 61
```

```
{0,65,34,20,8},                 // >        ASCII 62
{32,64,69,72,48},               // ?        ASCII 63
{38,73,77,65,62},               // @        ASCII 64
{31,36,68,36,31},               // A        ASCII 65
{127,73,73,73,54},              // B        ASCII 66
{62,65,65,65,34},               // C        ASCII 67
{127,65,65,34,28},              // D        ASCII 68
{127,73,73,65,65},              // E        ASCII 69
{127,72,72,72,64},              // F        ASCII 70
{62,65,65,69,38},               // G        ASCII 71
{127,8,8,8,127},                // H        ASCII 72
{0,65,127,65,0},                // I        ASCII 73
{2,1,1,1,126},                  // J        ASCII 74
{127,8,20,34,65},               // K        ASCII 75
{127,1,1,1,1},                  // L        ASCII 76
{127,32,16,32,127},             // M        ASCII 77
{127,32,16,8,127},              // N        ASCII 78
{62,65,65,65,62},               // O        ASCII 79
{127,72,72,72,48},              // P        ASCII 80
{62,65,69,66,61},               // Q        ASCII 81
{127,72,76,74,49},              // R        ASCII 82
{50,73,73,73,38},               // S        ASCII 83
{64,64,127,64,64},              // T        ASCII 84
{126,1,1,1,126},                // U        ASCII 85
{124,2,1,2,124},                // V        ASCII 86
{126,1,6,1,126},                // W        ASCII 87
{99,20,8,20,99},                // X        ASCII 88
{96,16,15,16,96},               // Y        ASCII 89
{67,69,73,81,97},               // Z        ASCII 90
{0,127,65,65,0},                // [        ASCII 91
{0,0,0,0,0},                    // \        ASCII 92
{0,65,65,127,0},                // ]        ASCII 93
{16,32,64,32,16},               // ^        ASCII 94
{1,1,1,1,1},                    // _        ASCII 95
{0,64,32,16,0},                 // `        ASCII 96
{2,21,21,21,15},                // a        ASCII 97
{127,5,9,9,6},                  // b        ASCII 98
{14,17,17,17,2},                // c        ASCII 99
{6,9,9,5,127},                  // d        ASCII 100
{14,21,21,21,12},               // e        ASCII 101
{8,63,72,64,32},                // f        ASCII 102
{24,37,37,37,62},               // g        ASCII 103
{127,8,16,16,15},               // h        ASCII 104
{0,0,47,0,0},                   // i        ASCII 105
{2,1,17,94,0},                  // j        ASCII 106
```

```
 {127,4,10,17,0},                    // k           ASCII 107
 {0,65,127,1,0},                     // l           ASCII 108
 {31,16,12,16,31},                   // m           ASCII 109
 {31,8,16,16,15},                    // n           ASCII 110
 {14,17,17,17,14},                   // o           ASCII 111
 {31,20,20,20,8},                    // p           ASCII 112
 {8,20,20,12,31},                    // q           ASCII 113
 {31,8,16,16,8},                     // r           ASCII 114
 {2,21,21,21,9},                     // s           ASCII 115
 {16,126,17,1,2},                    // t           ASCII 116
 {30,1,1,2,31},                      // u           ASCII 117
 {28,2,1,2,28},                      // v           ASCII 118
 {30,1,6,1,30},                      // w           ASCII 119
 {17,10,4,10,17},                    // x           ASCII 120
 {24,5,5,5,30},                      // y           ASCII 121
 {17,19,21,25,17},                   // z           ASCII 122
 {0,0,8,54,65},                      // {           ASCII 123
 {0,0,127,0,0},                      // |           ASCII 124
 {65,54,8,0,0},                      // }           ASCII 125/Z
};

void setup() {
 pinMode(latchPin2, OUTPUT);
 pinMode(clockPin2, OUTPUT);
 pinMode(dataPin2, OUTPUT);
 pinMode(psensor, INPUT);
 pinMode(proxy, INPUT);
 setupDispaly();

}
void setupDispaly() {
  Serial.begin(9600);

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3D for 128x64
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }

 .
  display.display();
  delay(2000); // Pause for 2 seconds


  display.clearDisplay();
```

```
  display.display();


  display.fillRect(0, 0, 128, 15, SSD1306_INVERSE);
  display.drawRect(110, 2, 16, 12, SSD1306_BLACK);
  display.setTextSize(1);
  display.setTextColor(SSD1306_BLACK);
  display.setCursor(113,4);
  display.println("OK");
  display.display();
  // Display Letter Board 3 rows 9 character in each row
  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  for (int j=0; j<3;j++){
    for (int i=0; i<9;i++){
       display.setCursor(i*12+2*i+1,j*16+17);
       display.println(Letters[j][i]);
       delay(10);
     display.display();
    }
  }


  display.fillRect(0, 16, 12, 16, SSD1306_INVERSE);
  display.display();

}

void Highlight_letter(int X, int X_Old, int Y, int Y_Old){



  display.fillRect(X*12+2*X, Y*16 +16, 12, 16, SSD1306_INVERSE);


  display.fillRect(X_Old*12+2*X_Old, Y_Old*16 +16, 12, 16, SSD1306_INVERSE);

  display.display();
}
void Selecttext() {
  Key_read =analogRead(Keyboard);
  if (Prev_Key_read>1000 and Key_read<1000){
    Key_pressed=true;
    if (Key_read<10 and Old_X>0) New_X=Old_X-1;
    if (Key_read>160 and Key_read<170 and Old_X<9) New_X=Old_X+1;
```

```
    if (Key_read>25 and Key_read<35 and Old_Y>-1) New_Y=Old_Y-1;
    if (Key_read>80 and Key_read<90 and  Old_Y<2 ) New_Y=Old_Y+1;
    if (Key_read>350 and Key_read<360) {
     if (New_Y!=-1){
       To_Transmit=To_Transmit + Letters[New_Y][New_X];
       To_Transmit_Length++;
       display.setTextSize(1);
       display.setCursor(3,1);
       display.setTextColor(BLACK     );
       display.fillRect(0, 0, 100, 15, SSD1306_WHITE);
       display.println(To_Transmit);
       display.display();
     }
     else{
      for (int i=1;i<9;i++) {
        display.fillRect(0, 0, 128, 15, SSD1306_INVERSE);
        delay(300);
        display.display();
      }
     }
    }

    if (New_Y==-1 and Old_Y==0){
        display.fillRect(110, 2, 16, 12, SSD1306_INVERSE);
        display.fillRect(Old_X*12+2*Old_X, Old_Y*16 +16, 12, 16,
SSD1306_INVERSE);
    }
    if (New_Y==0 and Old_Y==-1){
        display.fillRect(110, 2, 16, 12, SSD1306_INVERSE);
        display.fillRect(New_X*12+2*New_X, New_Y*16 +16, 12, 16,
SSD1306_INVERSE);
        Prev_Key_read=Key_read;
        Old_X=New_X;
        Old_Y=New_Y;;
    }
    if ((Old_X!=New_X or Old_Y!=New_Y) and Old_Y!=-1 ){
        if (New_Y!=-1 )Highlight_letter (New_X,Old_X,New_Y,Old_Y);
        Old_X=New_X;
        Old_Y=New_Y;
    }
  }

  display.display();
  Prev_Key_read=Key_read;
```

```
}


void RefreshDisplay()
{
 for (int row = 0; row < 8; row++)
 {
   int rowData = 1 << row;
   int rowBits = bitmap[row];

   digitalWrite(latchPin2, LOW);
   shiftOut(dataPin2, clockPin2, MSBFIRST, rowBits);
   shiftOut(dataPin2, clockPin2, MSBFIRST, rowData);
   digitalWrite(latchPin2, HIGH);
 }
}


void Plot(int x, int y, bool isOn)
{
 int row = y;
 int colBit = 1 << (7 - x);
 if (isOn)
   bitmap[row] |= colBit;
 else
   bitmap[row] &= ~colBit;
}


void XProcess()
{
 for (int charIndex = 0; charIndex < (sizeof(msg) - 1); charIndex++)
 {
   int alphabetIndex = msg[charIndex] - ' ';
   if (alphabetIndex < 0) alphabetIndex = 0;

   for (int row = 0; row < 8; row++)
   {
     bool isOn = bitRead(alphabets[alphabetIndex][row], 0) == 1;
     Plot(0, row, isOn); ly
   }
   for (int refreshCount = 0; refreshCount < scrollspeed; refreshCount++)
     RefreshDisplay();
 }
```

```
}

void loop() {

  Selecttext();
  msg[]=To_Transmit;
  if(digitalRead(psensor) == HIGH && digitalRead(proxy) == HIGH){
    XProcess();

  }

}
```