**Project Title: Remote Stepper Motor Controller System**

***Course title:*** *Microprocessors and Assembly Language Laboratory*

***Course code:*** *CSE-358*

*3rd Year 2nd Semester Examination 2023*

**Date of Submission**: **25 June, 2025**

****

**Submitted to-**

**Dr. MD. Golam Moazzam**

*Professor*

**Dr. Israt Jahan**

*Professor*

**Sabrina Sharmin**

*Associate Professor*

*Department of Computer Science & Engineering*

*Jahangirnagar University*

*Savar, Dhaka-1342*

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl** | Class Roll | Exam Roll | Name |
| **01** | **383** | **210903** | **Abdullah Nazmus-Sakib** |
| **02** | **391** | **210911** | **Khaled Mahmud Jon** |
| **03** | **379** | **210899** | **Shah Sultan** |
| **04** | **395** | **210915** | **Tawhidul Islam** |

**Contents**

**1. Introduction …..………………………………………………………………….01**

1.1 Background ..………...………………………………………………………01

1.2  Problem Statement ...…….....……..…………………………………………01

1.3Purpose ..…….……………………………………………………………….01

1.4 Objectives ……... ………………………………………………………..…..01

**2. Scope, Importance & Applications ..……………………………………………02**

2.1 Project Scope …………...…………………………………...……………….02

2.2 Traditional Methods of Liquid Level Measurement ………………...……....02

2.3 Importance of IoT Based Liquid Level Monitoring …………………………02

2.4 Real Life Application .……………………………………………………….03

**3. Circuit Design & Implementation ...……………………………………………03**

3.1 Used Components ……...…………………………………...……………….03

3.2 Circuit Diagram ……………………………………………………...……....04

3.3 Circuit Connection Overview …………………….…………………………04

3.4 Components Specifications ………………………………………………….05

4. **System Workflow ……………………………………………………………………07**

**5. Code Overview & Explanation …...……………………………………………07**

5.1 Used Compiler, Board & Library Function ….…...………...……………….07

5.2 Code Highlights………...………………………….………………...……....07

5.2.1EEPROM Integration …………………………………………………………………07

5.2.2 OLED Display …………………………………………………………………………08

5.2.3 Cycle Mode ……………………………………………………………………………08

5.2.4 Project Name and Contributors ……………………………………………………….09

5.2.5 Key Logics …………………………………………………………………………….09

5.3 Detailed breakdown of all Code…..………...…………...………………….…11

6. Screenshot …………………………………………………………………………14

**6. Conclusion ………………………………………………………………………..15**

**1. Introduction**

* 1. **Background**

Stepper motors are essential in applications requiring precise control over motion, such as CNC machines, robotics, and automation systems. The 28BYJ-48 stepper motor, paired with the ULN2003 driver, is a cost-effective solution for learning and experimentation in embedded systems. This project was developed as part of the CSE-358 Microprocessors and Assembly Language Laboratory course to explore hardware control using microcontrollers. By utilizing the ESP32, IR remote, and OLED display, we aimed to design a user-friendly system that enables remote control over motor speed, direction, and movement cycles. The integration of EEPROM allows persistent storage of settings, enhancing usability. This project not only reinforces embedded programming concepts but also introduces practical implementation of communication protocols, motor control logic, and user interface development.

**1.2 Problem Statement**

In many embedded and automation systems, controlling the movement of stepper motors typically requires manual input or complex setups involving wired controllers, which limits flexibility and user convenience. Beginners often struggle with implementing real-time motor control that is both interactive and intuitive. Additionally, feedback mechanisms such as status displays or persistent settings are usually absent in basic motor control systems. This project aims to solve these limitations by developing a wireless, remote-controlled stepper motor system using ESP32, IR remote, and an OLED display. The system should support directional control, speed adjustment, cycle-based movement, and pause/resume functionality. It must also provide real-time feedback via display and store the last used speed and direction using EEPROM. The goal is to make motor control both efficient and accessible for learning and small-scale automation applications.

**1.3 Purpose**

The purpose of this project is to design and implement a user-friendly stepper motor control system that allows remote operation using an IR remote and real-time status monitoring via an OLED display. By integrating an ESP32 microcontroller, the system enables wireless control over motor direction, speed, and cycle-based movements. It also stores user preferences using EEPROM, ensuring consistent behavior across restarts. This project serves as a practical learning platform for students to understand embedded systems, motor control, and human-machine interaction while providing a foundation for future enhancements like Wi-Fi-based control and smart automation features.

**1.4 Objectives**

The primary objective of this project is to build a remote-controlled stepper motor system using ESP32, enabling users to operate the motor easily with an IR remote. Specific goals include controlling the motor’s direction and speed, executing precise cycle-based movements, and displaying real-time feedback through a 0.96” OLED screen. Additional objectives involve storing user-defined speed and direction settings using EEPROM for persistent operation and enhancing user interaction through visual status indicators. This project aims to strengthen understanding of embedded systems, motor interfacing, and practical control applications, making it ideal for academic learning and future IoT-based development.

**2. Scope, Importance & Applications**

**2.1 Project Scope**

This project focuses on designing a remote-controlled stepper motor system using the ESP32 microcontroller, 28BYJ-48 motor, ULN2003 driver, IR remote, and OLED display. It allows users to control motor direction, speed, and specific cycle-based movement remotely with real-time visual feedback. The scope includes implementing EEPROM for persistent speed/direction memory and building a user-friendly interface without relying on physical buttons. While the current version uses IR communication, the design is scalable for future enhancements such as Wi-Fi/web control, limit switches, and precise angular movement, making it suitable for both academic demonstration and real-world automation use cases.

**2.2 Traditional Methods of Liquid Level Measurement**

Before the adoption of modern microcontrollers like the ESP32, traditional stepper motor control systems relied on wired or limited-range solutions using discrete hardware. These methods included:

* **Manual Switches and Buttons**: Basic control using physical toggle switches to change motor direction or start/stop actions.
* **Relay-Based Circuits**: Employed electromechanical relays to control motor phases, often bulky and less reliable.
* **555 Timer & Logic Gates**: Used to generate step pulses, requiring complex circuit design and lacking flexibility.
* **PLC Systems**: Programmable Logic Controllers provided robust control but were expensive and less accessible for small-scale or educational use.
* **Infrared with No Feedback**: IR remotes were sometimes used, but without any display or acknowledgment, making user interaction guess-based and error-prone.

These methods lacked programmability, real-time feedback, and scalability, which are now efficiently addressed using microcontrollers, sensors, and displays.

**2.3 Importance of IoT Based Liquid Level Monitoring**

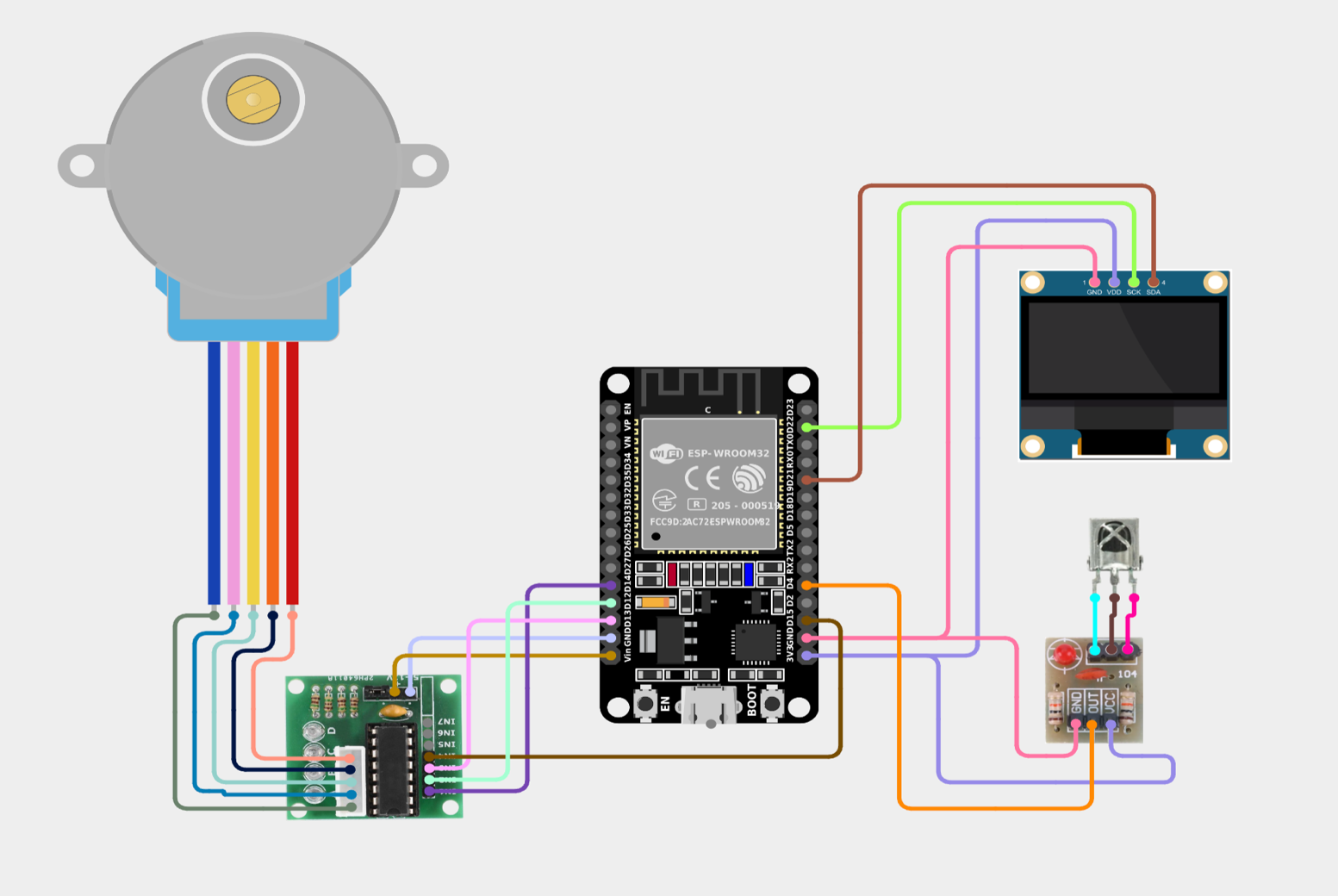
The integration of IoT with stepper motor control systems brings a significant advancement in precision, flexibility, and accessibility. Traditional systems lack scalability and real-time monitoring, while IoT-enabled controllers allow remote operation, data logging, and automation through wireless communication. Such systems can be accessed via mobile apps or web interfaces, enabling control from anywhere. This is crucial for applications in smart manufacturing, home automation, robotics, and agriculture where unattended and efficient control is essential. The addition of feedback mechanisms like OLED displays and cloud connectivity ensures better decision-making and error handling. IoT transforms basic motor control into an intelligent, adaptive, and user-friendly system.

* 1. **Real Life Applications**
  2. **Smart Home Automation**
     1. Automated window blinds, curtain control, or rotating solar panels using remote-controlled stepper motors.
  3. **3D Printers & CNC Machines**
     1. Precision control of motors for exact movements, used in additive manufacturing and engraving machines.
  4. **Robotics**
     1. Remote manipulation of robotic arms for pick-and-place tasks or joint control in hobby and industrial robots.
  5. **Agriculture & Irrigation**
     1. Controlling valves or adjusting solar panel angles on farms to improve efficiency and reduce manual labor.
  6. **Medical Equipment**
     1. Used in syringe pumps, prosthetics, or automated sample dispensers where fine, remote-controlled motion is required.
  7. **Security & Surveillance**
     1. Camera panning systems and automated locks that can be remotely adjusted using IR or wireless inputs.

**3. Circuit Design and Implementation**

**3.1 Used Components**

|  |  |
| --- | --- |
| Component | Description |
| ESP32 Dev Board | Core microcontroller with Wi-Fi and Bluetooth support |
| 28BYJ-48 Stepper Motor | Unipolar motor suitable for small automation |
| ULN2003 Motor Driver | Controls current to the stepper coils |
| IR Remote + Receiver Module | User interface for remote control |
| 0.96” OLED Display (SSD1306) | Displays motor status, speed, direction |
| Jumper Wires, Breadboard, Power Supply | For wiring and setup |

**3.2 Circuit Diagram**

**Figure:** Remote stepper motor controller system

* 1. **Circuit Connection Overview**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Component | Pin | Connected to (ESP32 GPIO) | Power Supply | Remarks |
| Stepper Motor IN1 | IN1 | GPIO 14 | From ULN2003 | Motor coil control |
| Stepper Motor IN2 | IN2 | GPIO 12 | From ULN2003 |  |
| **Stepper Motor IN3** | IN3 | GPIO 13 | From ULN2003 |  |
| **Stepper Motor IN4** | IN4 | GPIO 15 | From ULN2003 |  |
| **ULN2003 Driver Board** | GND/VCC | GND / External 5V | External 5V | Connect GND with ESP32 GND |
| IR Receiver | OUT | GPIO 4 | 3.3V (VCC) | Used for receiving IR commands |
|  | VCC | 3.3V |  |  |
|  | GND | GND |  |  |
| **OLED Display (0.96”)** | SDA | GPIO 21 | 3.3V | I2C Data Line |
|  | SCL | GPIO 22 | 3.3V | I2C Clock Line |
|  | VCC | 3.3V |  |  |
|  | GND | GND |  |  |

* 1. **Components Specifications**

**⚙️ ESP32 Specification**

|  |  |
| --- | --- |
| Feature | Details |
| Microcontroller | Xtensa® Dual-Core 32-bit LX6 microprocessor |
| Clock Speed | Up to 240 MHz |
| Operating Voltage | 3.0V – 3.3V |
| Input Voltage (USB) | 5V (via Micro USB) |
| Digital I/O Pins | 30 to 36 GPIO pins (varies by board) |
| Analog Input Pins (ADC) | 18 channels (12-bit SAR ADC) |
| Analog Output (DAC) | 2 channels (8-bit) |
| Flash Memory | 4 MB (can vary depending on module) |
| SRAM | 520 KB |
| Wi-Fi | 802.11 b/g/n (2.4 GHz) |
| Bluetooth | v4.2 BR/EDR and BLE |
| UART | 3 UART interfaces |
| SPI / I2C / I2S | Multiple interfaces supported |
| PWM | Available on most GPIO pins |
| Timers | Hardware timers and watchdog timers |
| Security | AES, SHA-2, RSA, ECC, Random Number Generator |
| Power Modes | Active, Modem-sleep, Light-sleep, Deep-sleep |
| Development Board Example | ESP32 DevKit v1 / NodeMCU-32S |
| Programming Environment | Arduino IDE, PlatformIO, MicroPython, ESP-IDF |
| USB to Serial | Built-in (usually CP2102 or CH340 chip) |

**🖥️ 0.96” OLED Display Specification (SSD1306)**

|  |  |
| --- | --- |
| Feature | Details |
| Display Type | OLED (Organic Light Emitting Diode) |
| Driver IC | SSD1306 |
| Screen Size | 0.96 inches |
| Resolution | 128 x 64 pixels |
| Color | Monochrome (usually White, Blue, or Yellow) |
| Interface | I2C (default) or SPI (some variants) |
| I2C Address | 0x3C (default for most modules) |
| Voltage Supply | 3.3V or 5V (compatible with both) |
| Current Consumption | < 20 mA |
| Viewing Angle | >160° |
| Operating Temperature | -40°C to +85°C |
| Communication Pins | SDA (Data), SCL (Clock) |
| Typical Use Cases | Status displays, clocks, microcontroller interfaces |
| Library Support | Adafruit SSD1306, U8g2, etc. (for Arduino, ESP32, etc.) |

### ⚙️ 28BYJ-48 Stepper Motor Specifications

|  |  |
| --- | --- |
| Feature | Details |
| Motor Type | Unipolar Stepper Motor |
| Rated Voltage | 5V DC |
| No. of Phases | 4 |
| Steps per Revolution | 2048 (for 1 full revolution in half-step mode) |
| Gear Ratio | 1:64 |
| Step Angle | 5.625° / 64 ≈ 0.0879° per step |
| Current per Phase | ~240 mA |
| Drive Method | Full-step / Half-step |
| Direction Control | Via sequence of pulses to coils |
| Applications | Robotics, automation, camera sliders, 3D printers |

### 🔌 ULN2003 Motor Driver Module Specifications

|  |  |
| --- | --- |
| Feature | Details |
| Driver IC | ULN2003A |
| Channels | 7 Darlington Transistor Arrays (only 4 used for motor) |
| Input Voltage | 5V – 12V DC |
| Output Current per Channel | 500 mA (max) |
| Logic Control Voltage | 3.3V or 5V (ESP32/Arduino compatible) |
| Input Pins | IN1, IN2, IN3, IN4 (to control 4 motor coils) |
| Output Pins | OUT1, OUT2, OUT3, OUT4 (connect to motor coils) |
| Indicators | 4 LEDs to indicate coil activation |
| Power Supply Pin | VCC and GND |
| Connector | 5-pin JST connector for 28BYJ-48 motor |

**5.** **System Workflow**

1. **System Boot-up**:

Displays project intro and contributor names.

Loads speed and direction settings from EEPROM.

1. **IR Command Reception**:

Uses IRremote library to receive command values.

Commands include direction control, speed adjustment, cycle count,

pause/resume.

1. **Motor Movement**:

Directional buttons (Forward/Backward) control continuous movement.

Number keys (0–9) allow selecting number of cycles (mapped to 1–10).

Uses AccelStepper library for smooth movement.

1. **Display Feedback**:

OLED shows current command, speed, direction, and motor status.

Includes smooth progress bar while executing multi-cycle movement.

1. **Persistent Settings**:

Saves speed and direction in EEPROM upon every valid command.

**Code Overview & Explanation**

**5.1 Used Compiler, Board & Library Function**

* Arduino IDE Compiler
* ESP32 Dev Module Board
* <IRremote.hpp>
* <AccelStepper.h>
* <Wire.h>
* <Adafruit\_GFX.h>
* <Adafruit\_SSD1306.h>
* <EEPROM.h>

**5.2 Code Highlights**

**5.2.1 EEPROM Integration**: Saves last speed and direction to avoid reset loss.

* // EEPROM Addresses
* #define EEPROM\_SPEED\_ADDR 0
* #define EEPROM\_DIR\_ADDR 4
* void saveToEEPROM() {
* EEPROM.writeInt(EEPROM\_SPEED\_ADDR, speed);
* EEPROM.writeBool(EEPROM\_DIR\_ADDR, previousDirectionForward);
* EEPROM.commit();
* }
* void loadFromEEPROM() {
* speed = EEPROM.readInt(EEPROM\_SPEED\_ADDR);
* previousDirectionForward = EEPROM.readBool(EEPROM\_DIR\_ADDR);
* if (speed < 100 || speed > 1000) speed = 300;

**5.2.2 OLED Display**: Provides real-time updates with command feedback and a progress bar.

1. #include <Wire.h>
2. #include <Adafruit\_GFX.h>
3. #include <Adafruit\_SSD1306.h>
4. #define SCREEN\_WIDTH 128 // OLED display width
5. #define SCREEN\_HEIGHT 64 // OLED display height
6. // Create display object (I2C address 0x3C is common)
7. Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire, -1);
8. void setup() {
9. Serial.begin(115200);
10. // Initialize OLED
11. if (!display.begin(SSD1306\_SWITCHCAPVCC, 0x3C)) {
12. Serial.println(F("SSD1306 allocation failed"));
13. for (;;); // Don't proceed, loop forever
14. }
15. delay(1000);
16. display.clearDisplay(); // Clear the buffer
17. // Display "Hello, ESP32!"
18. display.setTextSize(2);
19. display.setTextColor(SSD1306\_WHITE);
20. display.setCursor(0, 10);
21. display.println("Hello,");
22. display.setCursor(0, 30);
23. display.println("AbdullahRFA!");
24. display.display(); // Show on screen
25. }
26. void loop() {
27. // Nothing to do here
28. }
    * 1. **Cycle Mode**: User can press a number key (0 = 1 cycle, 1 = 2 cycles…) followed by Forward/Backward.

|  |
| --- |
| 1. void moveNCycle(int n, bool dir) { 2. stepper.setSpeed(dir ? abs(speed) : -abs(speed)); 3. int steps = 4096; // one cycle for 28BYJ-48 4. stepper.move(steps \* (dir ? n : -n)); 5. while (stepper.distanceToGo() != 0) { 6. stepper.run(); 7. } 8. // Display final movement summary 9. display.clearDisplay(); 10. display.setCursor(0, 0); 11. display.setTextSize(1); 12. display.print("Moved "); 13. display.print(n); 14. display.print(" Cycle "); 15. display.println(dir ? "Forward" : "Backward"); 16. display.display(); 17. delay(1000); 18. } |

* + 1. **Project Name and Contributors:** At the time of starting project it shows project name and the name of contributors

|  |
| --- |
| 1. void showProjectIntro() { 2. display.clearDisplay(); 3. display.setTextSize(1); 4. display.setCursor(0, 0); 5. display.println("Remote Stepper Motor"); 6. display.println("Controller System"); 7. display.display(); 8. delay(1500); // Show project title 9. display.clearDisplay(); 10. display.setCursor(0, 0); 11. display.setTextSize(1); 12. display.println("Presented by:"); 13. display.println("-------------"); 14. display.println("- Abdullah(383)"); 15. display.println("- Khaled(391)"); 16. display.println("- Shanto(379)"); 17. display.println("- Tawhid(395)"); 18. display.display(); 19. delay(2000); // Show contributor names 20. } |

* + 1. **Key logics:**

|  |  |
| --- | --- |
| 1. void loop() { 2. if (IrReceiver.decode()) { 3. int command = IrReceiver.decodedIRData.command; 4. Serial.print("IR Command: "); Serial.println(command); 5. // showCommandFeedback(command,dir); 6. if (command >= 0 && command <= 9) { 7. stepRequest = command + 1; // Consider 0 as 1, 1 as 2, ..., 9 as 10 8. cycleAwaitDirection = true; 9. displayStatus("Press F/B for Cycle"); 10. } 11. else if (cycleAwaitDirection && (command == forward || command == backword)) { 12. bool dir = (command == forward); 13. showCommandFeedback(stepRequest, dir); // Show detailed feedback 14. moveNCycle(stepRequest, dir); // Just move requested cycles once 15. cycleAwaitDirection = false; 16. stepRequest = 0; 17. }else { 18. switch (command) { 19. case 17: 20. moveForward = true; 21. moveBackward = false; 22. motorActive = true; 23. previousDirectionForward = true; 24. idleToggleState = false; 25. displayStatus("Moving Forward"); 26. break; 27. case 16: 28. moveBackward = true; 29. moveForward = false; 30. motorActive = true; 31. previousDirectionForward = false; 32. idleToggleState = false; 33. displayStatus("Moving Backward"); 34. break; 35. case 19: 36. speed += 100; 37. if (speed > 1000) speed = 1000; 38. stepper.setSpeed(speed); 39. displayStatus("Speed Increased"); 40. break; 41. case 18: 42. speed -= 100; 43. if (speed < 100) speed = 100; 44. stepper.setSpeed(speed); 45. displayStatus("Speed Decreased"); 46. break; | case 11:  moveForward = false;  moveBackward = false;  motorActive = false;  idleToggleState = false;  stepper.stop();  displayStatus("Motor Stopped");  break;  case 28:  if (!idleToggleState) {  motorActive = false;  idleToggleState = true;  displayStatus("Paused (Idle)");  } else {  motorActive = true;  if (previousDirectionForward) {  moveForward = true;  moveBackward = false;  } else {  moveBackward = true;  moveForward = false;  }  idleToggleState = false;  displayStatus("Resumed from Idle");  }  break;  default:  displayStatus("Unknown Cmd");  break;  }  }  saveToEEPROM();  IrReceiver.resume();  delay(300);  }  if (motorActive) {  if (moveForward) {  stepper.setSpeed(abs(speed));  stepper.runSpeed();  } else if (moveBackward) {  stepper.setSpeed(-abs(speed));  stepper.runSpeed();  }  }  } |

**5.3 Detailed breakdown of all IR remote button functionalities** for Our **Remote Stepper Motor Controller System** project, based on Our final Arduino code:

**🔘 IR Remote Button Functionality**

Each button on the IR remote triggers a specific action on the ESP32-powered stepper motor controller. Below are all key mappings and their corresponding functionalities:

**🔢 Number Buttons (0–9)**

**Function:** Select number of full motor cycles to move.

* **Behavior:**
  + Pressing 0 selects **1 cycle**.
  + Pressing 1 selects **2 cycles**.
  + …
  + Pressing 9 selects **10 cycles**.
* **Display Feedback:** Shows *“Press F/B for Cycle”* on OLED.
* **Next Step Required:** After selecting a number, you must press either the **Forward** or **Backward** button to execute the movement.

**🔼 Forward Button (forward = 17)**

* **Function 1:**

If a number was selected previously:

→ Moves the motor that many **cycles forward** (clockwise).

→ Shows a **progress bar** during movement.

→ Displays *“Moved X Cycle Forward”* upon completion.

* **Function 2:**

If no number was selected:

→ Starts **continuous forward motion** of the stepper motor.

→ Displays *“Moving Forward”* on OLED.

**🔽 Backward Button (backword = 16)**

* **Function 1:**

If a number was selected:

→ Moves the motor that many **cycles backward** (counterclockwise).

→ Shows a **progress bar** during movement.

→ Displays *“Moved X Cycle Backward”* upon completion.

* **Function 2:**

If no number was selected:

→ Starts **continuous backward motion** of the stepper motor.

→ Displays *“Moving Backward”* on OLED.

**⏹️ Stop Button (stopMotor = 11)**

* **Function:**

Immediately **stops the motor** whether it’s running continuously or performing a cycle.

* **Effects:**
  + motorActive = false, halts motor.
  + Shows *“Motor Stopped”* on OLED.

**⏸️ Pause / Resume Button (idlePosition = 28)**

* **Function:**

**Toggles pause/resume** of motor movement.

* **First Press:**
  + Pauses any ongoing motion.
  + Shows *“Paused (Idle)”* on OLED.
* **Second Press:**
  + Resumes motion in the **previous direction**.
  + Shows *“Resumed from Idle”* on OLED.

**➕ Speed Increase (speed\_increase = 19)**

* **Function:**

Increases the motor speed by 100 steps/sec.

* **Limits:**
  + Maximum speed is capped at 1000.
* **Display Feedback:**

Shows *“Speed Increased”* with current speed value.

**➖ Speed Decrease (speed\_decrease = 18)**

* **Function:**

Decreases the motor speed by 100 steps/sec.

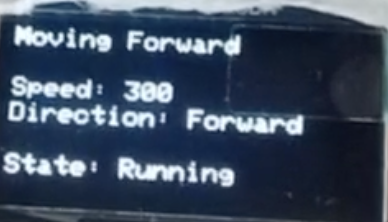
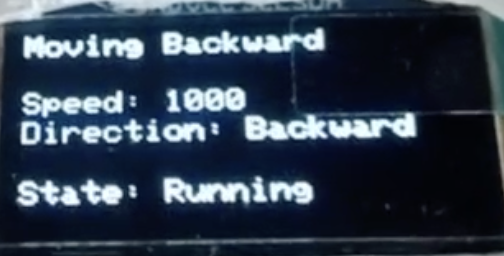
* **Limits:**
  + Minimum speed is set to 100.
* **Display Feedback:**

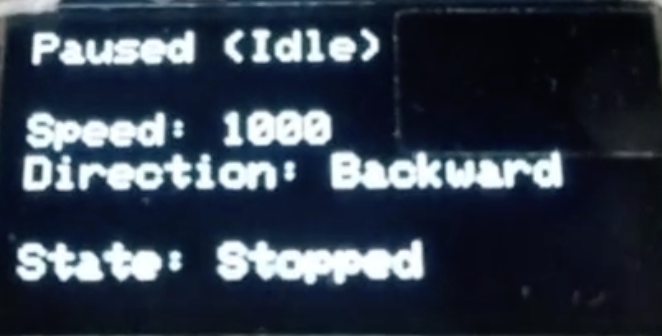
Shows *“Speed Decreased”* with current speed value.

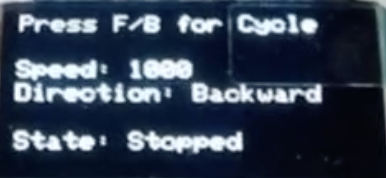
**📋 Summary Table**

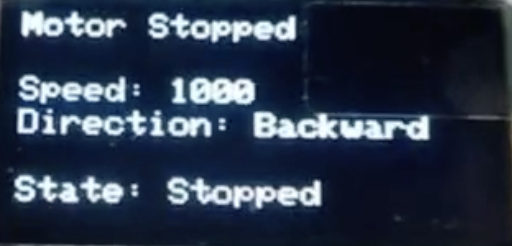
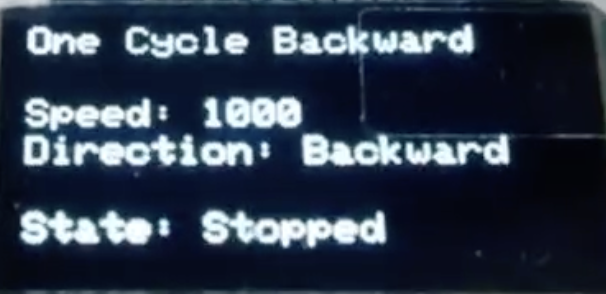
|  |  |  |
| --- | --- | --- |
| Button | Decimal Code | Functionality |
| 0–9 | 0–9 | Select number of cycles (0=1 cycle, 9=10 cycles) |
| Forward | 17 | Move forward (continuous or cycle) |
| Backward | 16 | Move backward (continuous or cycle) |
| Stop | 11 | Stop all motor activity |
| Pause/Resume | 28 | Pause or resume current motor movement |
| Speed + | 19 | Increase speed by 100 steps/sec |
| Speed - | 18 | Decrease speed by 100 steps/sec |

**13. Screenshots / Demo**









### 🧾 Conclusion

The **Remote Stepper Motor Controller System** successfully demonstrates an efficient and user-friendly method for controlling a stepper motor using an IR remote and OLED feedback, powered by the ESP32 microcontroller. This project showcases the integration of embedded hardware and software for real-time control, persistent settings via EEPROM, and interactive feedback through display. Users can easily operate the motor for forward/backward movement, control speed, and execute precise cycle-based rotations with visual updates. The system is scalable, reliable, and serves as a foundation for more advanced IoT and automation applications. This project not only fulfills academic objectives but also lays the groundwork for future enhancements like Wi-Fi/web control, mobile integration, and industrial usage.