A

project report

on

RFID DOOR LOCKING SYSTEM USING STM32F411RE

submitted to

AP IIIT RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

In partial fullfillment of the requirement of the degree

BACHELOR OF TECHNOLOGY IN ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by:

R.PAVANI R190452

M.NEERAJA R190377

Under the Guidance of

MR.N.MOHAN RAJU

- ***ASSISTANT PROFESSOR***

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

RKVALLEY,Vempalli(M), Kadapa(Dist), Andhra Pradesh(S),516330.

(2024-2025)

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE AND TECHNOLOGIES
RK VALLEY, KADAPA - 516330

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

2023-2024

CERTIFICATE

This is to certify that the study entitled on " RFID DOOR LOCKING SYSTEM USING STM32"bonafide work done and submitted by

R.PAVANI R190452

M.NEERAJA R190377

for the partial fulfillment of the requirements for the completion of B.Tech Degree, ELECTRONICS AND COMMUNICATION ENGINEERING, RGUKT RK Valley.

Guide	***Head of the Department***
MR.N.MOHAN RAJU	MR.ARUN KUMAR REDDY
Assistant Professor	Assistant Professor
RGUKT, RK Valley	RGUKT, RK Valley
Kadapa-516330	Kadapa-516330

External viva-voice exam held on-_____

INTERNAL EXAMINER EXTERNAL EXAMINER

DECLARATION

I hereby declare that the study entitled “RFID DOOR LOCKING SYSTEM USING STM32F411RE” submitted to the Department of ELECTRONICS AND COMMUNICATION ENGINEERING in partial fulfilment of requirements for the completion of the degree of BACHELOR OF TECHNOLOGY. This study is the result of my effort and that it has not been submitted to any other University or institution for the award of any Degree or Diploma other than specified above .

R.PAVANI R190452

M.NEERAJA R190377

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE AND TECHNOLOGIES
RK VALLEY, KADAPA - 516330

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ACKNOWLEDGEMENT

We would like to express our deepest and heart full thanks to **Dr. A.V.S.S.KUMARA SWAMY GUPTA, DIRECTOR**, **RGUKT, RK VALLEY**, **Mr. ARUN KUMAR REDDY,** Head of the Department, **ELECTRONICS AND COMMUNICATION ENGINEERING,** for his kind help and encouragement during the course of our study and in the successful completion of the project.

We are thankful to our guide **MR.N.MOHAN RAJU, Assistant Professor, Department of ELECTRONICS AND COMMUNICATION ENGINEERING** for his valuable guidance and encouragement. His helping attitude and suggestions have helped us in the successful completion of the project.

R.PAVANI R190452
M.NEERAJA R190377

INDEX

CHAPTER – 1

1.1 INTRODUCTION

In the modern era, digital security has become a vital component of both residential and commercial infrastructure. Traditional key-based door locking mechanisms, while still commonly used, pose several security and convenience challenges. Keys can be lost, stolen, duplicated, or misused, leading to potential breaches in safety and unauthorized access. As a result, there has been a noticeable shift toward adopting smarter, more secure electronic access control systems that are both user-friendly and robust.

One of the most practical and widely adopted solutions in this domain is the RFID (Radio Frequency Identification) based door locking system . RFID technology allows for contactless user identification using RFID cards or tags, eliminating the need for physical keys. These systems enhance convenience while significantly improving security by allowing only authorized individuals to access a restricted area.

This project focuses on the development and implementation of an RFID-based door locking system using the STM32F411RE microcontroller. The STM32F411RE is part of STMicroelectronics' STM32 family and is built on the ARM Cortex-M4 architecture, offering high processing power, efficiency, and a wide range of hardware interfaces. It is an excellent choice for embedded system applications requiring fast performance and flexible interfacing capabilities.

At the core of the system is the RC522 RFID reader module, which reads unique identifiers (UIDs) from RFID cards or tags presented to it. The STM32F411RE microcontroller receives the UID data via the SPI communication protocol and compares it against a predefined list of authorized UIDs stored in its memory. If a match is found, the system deems the user as authenticated.

Upon successful authentication, the microcontroller sends signals to servo motors that are connected to the door mechanism. These servo motors physically lock or unlock the door depending on the user's access rights. The current status of the system—such as “Access Granted” or “Access Denied” is displayed using a 16x2 I2C LCD module, allowing the user to receive immediate visual feedback.

What sets this project apart from many others is its cloud-based monitoring capability. Rather than using a hardware-based Wi-Fi module like the ESP8266 for network communication, this system utilizes a Python program running on a connected PC. This Python script reads the access log data from the STM32F411RE via serial UART communication, processes it, and uploads it to Adafruit IO, an Internet of Things (IoT) platform that supports real-time data visualization and logging.

By leveraging Adafruit IO, the system offers remote monitoring features without embedding Wi-Fi connectivity directly into the microcontroller setup. This approach simplifies the overall design and is particularly advantageous for educational and prototyping environments, where hardware complexity should be minimized without sacrificing functionality.

In summary, this RFID-based door locking system showcases an effective blend of embedded systems design, automation, and IoT integration. It not only provides enhanced security through RFID authentication but also enables real-time access tracking through a user-friendly cloud platform. The use of STM32F411RE ensures high performance and reliable operation, while the Python-driven data pipeline adds flexibility and scalability for future enhancements. This project is an excellent example of how modern technology can be utilized to create secure, smart, and connected systems for everyday applications

Fig:Door lock

Fig: RFID Technology

1.2 OBJECTIVES OF THE PROJECT

1. **To develop a secure and contactless door locking system**
   - Implement RFID technology to allow access control without physical keys, reducing the risk of unauthorized duplication or loss.

2. **To use the STM32F411RE microcontroller for system control**
   - Utilize the high-performance ARM Cortex-M4-based STM32F411RE to manage RFID data, servo control, and display output.

3. **To integrate the RC522 RFID module for user authentication**
   - Interface the RC522 module with STM32 via SPI to read RFID tags and verify access credentials.

4. **To control the locking mechanism using servo motors**
   - Use servo motors to physically lock and unlock the door based on authentication results, offering a simple and reliable mechanical interface.

5. **To display system status using a 16x2 I2C LCD**
   - Provide real-time feedback to users by displaying messages such as "Access Granted" or "Access Denied" on the LCD.

6. **To log access events using a Python script on a PC**
   - Create a Python program that communicates with STM32 via UART to collect data such as user IDs and timestamps.

7. **To upload and monitor access logs on Adafruit IO**
   - Use the Python program to send access data to Adafruit IO, enabling remote monitoring, visualization, and storage of event history.

8. **To avoid the use of Wi-Fi modules for simplicity and cost-effectiveness**
   - Achieve IoT functionality without directly embedding a Wi-Fi module like ESP8266, making the system easier to build and debug.

9. **To create a scalable and modular system**
   - Design the system in such a way that more features like biometric sensors, keypad entry, or mobile control can be added in the future.

10. **To promote learning and application of embedded systems and IoT**
    - Serve as an educational project that combines embedded programming, sensor interfacing, hardware control, and cloud-based data integration.

1.3 REAL TIME INCIDENTS:

Fig:Acessing Hotel Room

Fig:Acessing Sever Rooms Or Office

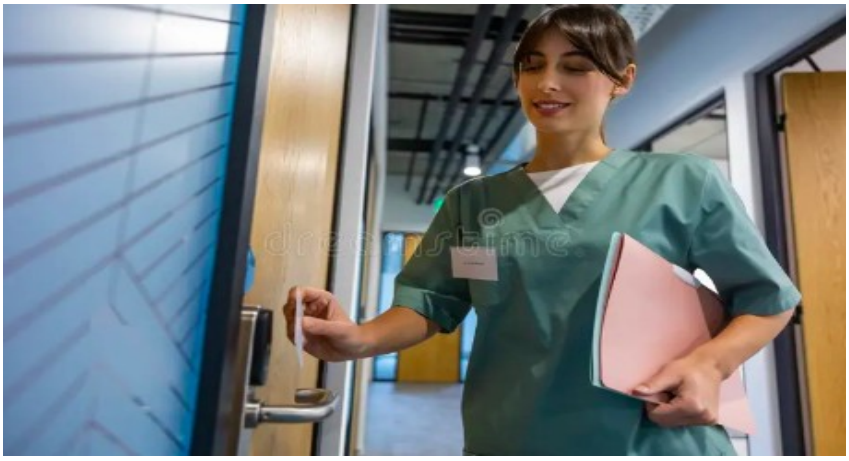


Fig:Acessing laborateries Or Medical Room

Real-Time Incidents Highlighting the Need for RFID Door Locking Systems

1. Unauthorized Entry in Office Buildings

- **Incident:** In several cases, former employees have used duplicate keys or unrevoked access cards to enter office premises after termination.
- **RFID Benefit:** RFID-based systems can instantly revoke access by removing the UID from the authorized list, making it impossible to re-enter without administrative approval.

2. Hotel Room Key Misuse

- **Incident:** Guests in some hotels have accidentally (or deliberately) accessed other rooms due to poor key management or manual errors in assigning keys.
- **RFID Benefit:** Each RFID card is uniquely programmed and mapped to a specific room, significantly reducing the chances of such errors. Access logs can also help track usage.

3. Student Hostel Intrusion

- **Incident:** In many student accommodations, unauthorized outsiders have been caught entering hostels by tailgating or using borrowed keys.
- **RFID Benefit:** RFID systems log every entry, providing timestamps and card IDs. Entry can be restricted to specific timeframes, and non-residents can be easily identified.

4. Theft in Apartment Complexes

- **Incident:** Thieves have broken into residential buildings using master keys or during maintenance hours when entry is unmonitored.
- **RFID Benefit:** RFID access can be limited to residents and logged for auditing. Combined with cloud monitoring (e.g., Adafruit IO), suspicious access patterns can be flagged.

5. Server Room Breaches in IT Firms

- **Incident:** Unauthorized access to server rooms has led to data theft or tampering in some companies.
- **RFID Benefit:** Using RFID access and real-time logging, administrators can track exactly who accessed the room and when—adding an additional layer of security.

6. School Safety Concerns

- **Incident:** In a few instances, unknown individuals have entered school premises, raising concerns about student safety.
- **RFID Benefit:** Schools can implement RFID systems for staff and students. This not only restricts access but also keeps a digital record of everyone's presence inside the campus.

CHAPTER – 2

COMPONENTS:

1By **using this below mentioned components we design the circuit for RFID Door locking sytsem using stm32f411re:**

HARDWARE:

1.STM32-F411RE

2.RFID RC522

3.LCD Display

4.Servo motors

5.Jumper wires

6.Breadboard

SOFTWARE:

1.Arduino IDE

2.IDLE(using python)

3.Ada fruit IO

HARDWARE COMPONENTS:

2.1 STM32-F411RE

Fig : STM32-411RE

The **STM32F411RE** is a high-performance microcontroller from the **STM32F4 series** by **STMicroelectronics**, built around the **ARM Cortex-M4 core**. It offers powerful processing capabilities, multiple communication interfaces, and rich peripheral support, making it ideal for embedded systems, robotics, IoT projects, and automation.

Core Feature:

Feature	Description
Core	ARM® Cortex®-M4 with FPU (Floating Point Unit)
Operating Frequency	Up to **100 MHz**
Flash Memory	512 KB
SRAM	128 KB
Voltage Range	1.7 V to 3.6 V
Package	LQFP64 (on Nucleo-F411RE board)

Key Peripherals and Interfaces

- **GPIO Pins**:Up to 50 general-purpose I/O pins
- **Timers**:Advanced-control, general-purpose, and basic timers
- **ADC**:12-bit ADC with up to 16 channels
- **USART/UART:**3 USART + 2 UART ports (ideal for serial communication with RFID module and PC)
- **SPI Interfaces**:3 SPI modules (used to interface with RFID RC522)
- **I²C Interfaces:**3 I²C buses (used for LCD display)
- **DMA**:Direct memory access controller for efficient data handling
- **PWM Output:**Multiple PWM channels (perfect for controlling servo motors)

Development Board – Nucleo-F411RE

Most often, STM32F411RE is used with the Nucleo-F411RE development board, which features:

- ST-LINK/V2 debugger for programming and debugging
- Arduino Uno R3 headers for easy shield compatibility
- Mbed-enabled platform for fast prototyping
- Micro-USB connector for power and communication

Why STM32F411RE For this project?

- Powerful processing to handle RFID authentication logic and multitasking
- Multiple serial ports (UART, SPI, I2C) to connect RFID reader, LCD, and PC simultaneously
- Fast PWM support for precise control of servo motors
- Low power consumption ideal for battery-powered door systems
- Community support and plenty of resources/documentation available

2.2 RFID - RC522

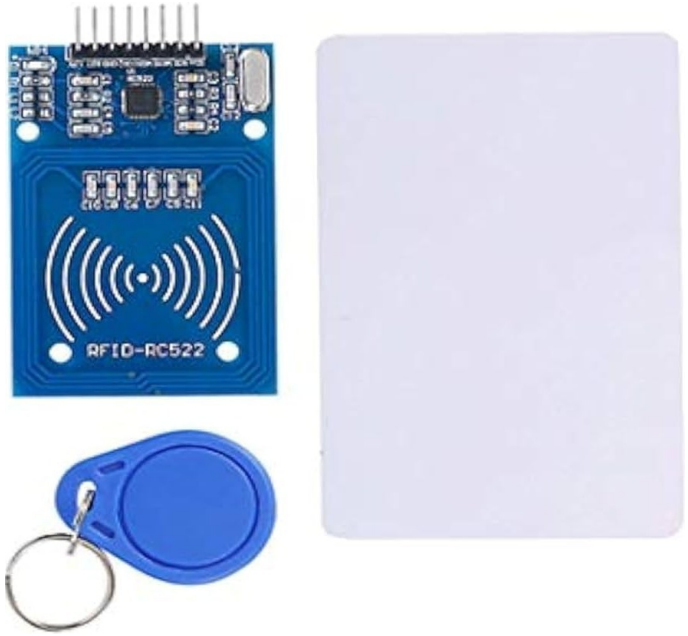


Fig :RFID Tag and Card

Fig :RFID Pin Configuration

Overview of RFID RC522:

The RFID RC522 is a low-cost and efficient RFID reader/writer module designed for contactless communication at 13.56 MHz. It is based on the MFRC522 chip developed by NXP Semiconductors and supports the ISO/IEC 14443A standard, making it compatible with a variety of RFID cards such as MIFARE 1K, 4K, and Ultralight. The module is widely used in applications such as electronic access control, attendance systems, identity verification, and automation. It communicates primarily via the SPI interface but can also support I2C and UART, making it versatile for integration with various microcontrollers like STM32, Arduino, and Raspberry Pi. The RC522 operates at 3.3V and includes an integrated antenna, capable of reading RFID tags within a short range of 2 to 5 cm. With features like anti-collision detection, fast communication, and built-in data encryption support, the RC522 provides a secure and reliable way to implement RFID-based authentication in embedded projects.

Role and Importance of the RFID RC522 in the Project

- **User Authentication**: The RFID RC522 serves as the primary component for identifying and authenticating users. It reads the unique ID (UID) from RFID cards or tags and sends this information to the microcontroller (STM32F411RE) for verification.
- **Contactless Access Control**: It enables a secure, contactless method for users to unlock the door, reducing the need for physical keys or touch-based systems, which enhances convenience and hygiene.
- **Data Collection for Monitoring**: By capturing and transmitting user access data, the RC522 helps in logging entry events. This data can be sent via a connected Python program to platforms like Adafruit IO for real-time monitoring.
- **Integration with Microcontroller**: The RC522 communicates with the STM32F411RE via the SPI interface, ensuring fast and reliable data exchange. This makes it easy to process access logic in real time.
- **Cost-Effective Security Solution**: The RC522 module is affordable, compact, and easy to use, making it an ideal choice for smart door lock systems, especially in educational or prototype-level applications.
- **Supports Multiple Users**: With support for multiple tag types and anti-collision features, the RC522 can efficiently handle authentication for several users in shared or public access scenarios.

Example Setup of a RFID RC522 in the System:

In the system setup, the RFID RC522 module is integrated with the STM32F411RE microcontroller using the SPI interface for fast data communication. The RC522 is connected to the STM32F411RE as follows: the SDA (SS) pin connects to PA4 (SPI1_NSS), the SCK pin connects to PA5 (SPI1_SCK), the MOSI pin to PA7 (SPI1_MOSI), and the MISO pin to PA6 (SPI1_MISO). The RST pin of the RC522 is connected to a general-purpose GPIO pin (e.g., PB1), while GND and 3.3V are connected to the appropriate pins of the STM32 board. When an RFID tag is placed near the RC522, it reads the unique UID of the tag and sends it to the STM32 for authentication. The STM32 then compares the UID to a list of authorized tags; if the UID matches, it triggers a servo motor to unlock the door and displays "Access Granted" on a 16x2 I2C LCD. If the UID is not authorized, "Access Denied" is shown on the LCD. The system also logs access attempts and sends this data to a PC via UART, where a Python script handles the data and uploads it to Adafruit IO for real-time monitoring. The RC522 operates on 3.3V, ensuring compatibility with the STM32F411RE's logic levels. This setup provides a secure, contactless, and efficient RFID-based access control system.

2.3 LCD DISPLAY

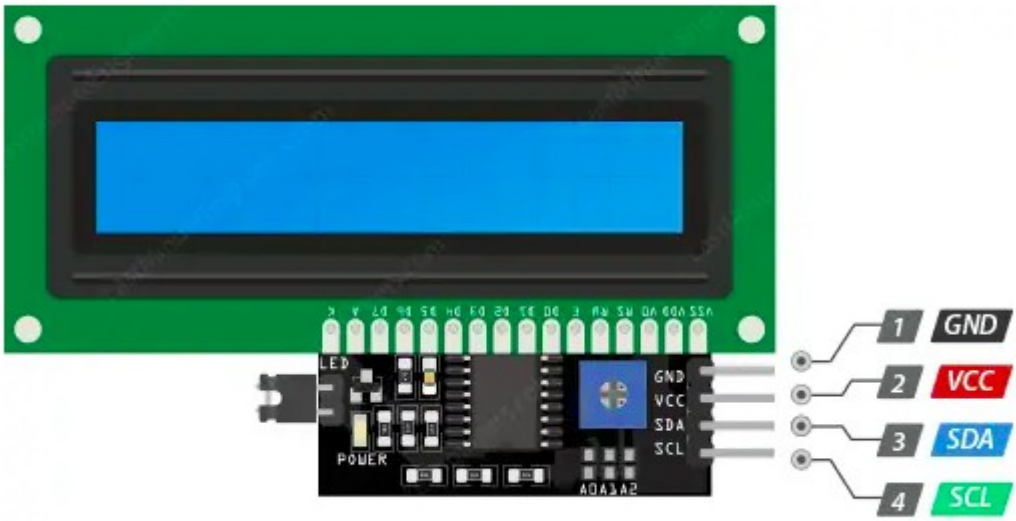


Fig: LCD display

The **LCD I2C display** is a 16x2 screen that combines a standard **LCD** with an **I2C interface**, allowing easy communication with microcontrollers like **STM32** and **Arduino**. It reduces wiring complexity by using only two data lines, **SDA** and **SCL**, for communication. This display can show up to **16 characters per row** and **2 rows** of text, making it ideal for status updates, feedback, and system monitoring in embedded projects. It offers features like adjustable contrast and brightness for better readability. The I2C interface also allows connecting multiple devices on the same bus, saving valuable I/O pins.

How a LCD Display Works

An LCD (liquid crystal display) works by using liquid crystals to control the passage of light and create visible characters or images. It consists of several layers including a backlight, polarizing filters, and a liquid crystal layer. The backlight provides constant illumination, while the polarizing filters and liquid crystals determine how much light is visible on the screen. When an electric current is applied to the liquid crystals, they change their orientation, allowing or blocking light from passing through specific areas. This is how pixels are turned on or off to form characters. In the case of an LCD with I2C interface, communication with a microcontroller is simplified using just two wires (SDA and SCL), making it easier to control text and display functions using minimal pins.

Types of LCD Display:

1. **Character LCD**
   - Displays only text in a fixed number of rows and columns (e.g., 16x2, 20x4).
   - Commonly used in microcontroller projects for simple output.

2. **Graphical LCD (GLCD)**
   - Can display images, custom characters, and graphics.
   - Examples include 128x64 and 240x128 pixel displays.

3. **Segment LCD**
   - Displays fixed segments (like digits or symbols), used in devices like calculators and digital clocks.
   - Common types include 7-segment and 14-segment displays.

4. **TFT LCD (Thin Film Transistor)**
   - A full-color LCD used for displaying high-resolution graphics and images.
   - Often used in smartphones, tablets, and advanced embedded systems.

5. **OLED (Organic Light Emitting Diode)**
   - Although technically different from traditional LCDs, it's often grouped similarly.
   - Offers better contrast and viewing angles, commonly used in modern displays.

6. **IPS LCD (In-Plane Switching)**
   - A type of TFT LCD that provides improved color accuracy and wider viewing angles.
   - Used in higher-end display panels like those in phones and monitors.

LCD Display Configurations

LCD Display support different configurations for various applications:

LCD displays come in various configurations to suit different project needs and environments. For simple applications, **character LCDs** like 16x2 or 20x4 are commonly used to display text, such as sensor readings or system status. For more complex interfaces, **graphical LCDs** offer pixel-based control, allowing the display of shapes, images, and custom fonts. **Segment LCDs** are ideal for fixed numerical or symbol-based outputs, such as in digital clocks, meters, or calculators. In applications requiring rich visuals, **TFT and IPS LCDs** provide high-resolution, full-color displays for user interfaces in consumer electronics, dashboards, and portable devices. Additionally, these LCDs can interface through **parallel or I2C/SPI protocols**, offering flexibility based on the number of I/O pins and performance requirements. This wide range of configurations makes LCDs highly adaptable for educational, industrial, and commercial uses.

Common Specifications of LCD Display

1. **Display Size** – Common sizes include 16x2, 20x4 for character LCDs, and 128x64 or 240x128 pixels for graphical LCDs.
2. **Interface Type**– Interfaces include parallel (typically 16-pin) or serial protocols like I2C and SPI for easier wiring.
3.**Operating Voltage**– Usually operates at 5V or 3.3V depending on the model.
4.**Backlight** – Includes an LED backlight for better visibility in low light; may be white, blue, or green.
5.**Character Format** – Each character is typically made of a 5x8 dot matrix.
6.**Viewing Angle** – Specifies the angle range within which the display is readable (e.g., 45°, 90°, etc.).
7.**Operating Temperature**– Most LCDs operate between -20°C to +70°C, suitable for standard environments.
8.**Controller IC**– Commonly uses the HD44780 controller for character LCDs or ST7920 for graphical LCDs.
9.**Contrast Adjustment–** Controlled via a potentiometer or software command for optimal text visibility.
10.**Power Consumption**– Low power usage, ideal for battery-operated or portable systems.

Applications of LCD Display

1. Embedded systems – Used to display real-time data, menus, and system status in microcontroller-based projects.
2. Consumer electronics – Found in calculators, digital clocks, remote controls, and household appliances.
3. Industrial equipment – Used in machine interfaces (HMIs), process monitoring, and control panels.
4. Medical devices – Displays vital signs, readings, and settings in devices like blood pressure monitors and ECG machines.
5. Automotive displays – Shows speed, fuel level, temperature, and other indicators in vehicle dashboards.
6. Communication devices – Used in landline phones, walkie-talkies, and other communication hardware.
7. Point of sale (POS) systems – Displays transaction details in cash registers and billing systems.
8. Security systems – Shows status messages, keypad inputs, and alerts in access control and alarm systems.
9. Educational kits and DIY projects – Popular among students and hobbyists for learning and prototyping.
10. IoT devices – Displays sensor data, network status, or control menus in smart home and connected systems.

2.4 SERVO MOTORS

Fig: Servo Motor

A servo motor is a small, precise motor used to control angular position with great accuracy. It includes a DC motor, gears, a position sensor, and a control circuit. The motor moves based on a PWM signal, which tells it the angle to rotate to, usually between 0 and 180 degrees. It is commonly used in robotics, automation, and embedded systems for tasks like controlling levers, arms, or locks. Servo motors are easy to control with microcontrollers like STM32 and respond quickly to input changes. Their compact size and reliability make them ideal for precise mechanical movements.

Basic Working Principle of a Servo Motor

The basic working principle of a servo motor is based on closed-loop control. It uses a control signal, typically a PWM (pulse width modulation) signal, to determine the desired position of the motor shaft. Inside the servo, a position sensor (usually a potentiometer) continuously monitors the shaft's angle. The internal control circuit compares the current position with the desired position. If there's any difference (error), it adjusts the motor’s rotation to correct it. This feedback loop continues until the motor reaches the target position. This mechanism allows the servo to provide precise and accurate control over angular motion, making it ideal for applications requiring controlled movement.

Key Components of a Servo Motor

The key components of a servo motor are:

1. **DC Motor**: The core component that provides rotational motion. The motor drives the shaft to rotate to the desired position based on the input signal.
2.**Gearbox**: A set of gears that reduces the speed of the motor and increases torque. This allows the servo to move slowly but with higher force, providing precise control.
3.**Position Sensor (Potentiometer):** A feedback device that continuously measures the current position of the motor's shaft and sends this information to the control circuit.
4.**Control Circuit:** The electronics responsible for interpreting the input signal (typically a PWM signal) and adjusting the motor's movement accordingly to achieve the target position.
5.**Housing**: The outer shell that contains and protects the internal components of the servo motor, such as the motor, gears, and electronics.

Types of Servo Motors

1.**AC Servo Motors:**
- These motors are powered by alternating current (AC) and are commonly used in industrial and high-performance applications.
- They offer high efficiency, smooth motion, and can operate at high speeds.
- AC servos are typically used in robotics, CNC machines, and automated manufacturing processes.

2.**DC Servo Motors:**
- Powered by direct current (DC), these motors are simpler and generally used in applications requiring less complexity.
- DC servo motors are ideal for low-cost, low-power applications such as hobby robots, small machines, and automation systems.
- They have good torque and speed control.

3.**Permanent Magnet Servo Motors (PMDC):**
- These motors use a permanent magnet for the rotor, which provides a stable magnetic field.
- They are commonly used in low to medium power applications where precise control and torque are needed.
- PMDC motors are efficient, have good responsiveness, and are used in applications like RC vehicles, camera systems, and small machinery.

Servo Motor Characteristics

1. **Torque:**
- The amount of rotational force a servo motor can generate. Torque is typically specified in kg-cm or Nm (Newton meters). Higher torque is required for moving heavier loads or applications with more resistance.

2.**Speed:**
- The rate at which the servo motor can rotate, usually measured in degrees per second or rpm (revolutions per minute). Higher speeds are necessary for applications needing fast response times.

3.**Precision/Resolution**:

- Servo motors are known for their high precision , meaning they can accurately reach and maintain specific positions. This precision is typically expressed in degrees (e.g., a motor may be able to rotate in 1° increments).

4.**Control Type:**

- Servo motors are controlled through a feedback loop , typically using PWM (pulse width modulation) signals to dictate the desired position, speed, or torque.

5.**Rotation Range:**

- Most hobby servos can rotate within a range of 0 to 180 degrees. However, certain servos (especially continuous rotation types) can rotate indefinitely.

6.**Voltage:**

- Servo motors operate within a specific voltage range, typically between 4.8V to 6V for hobby servos. Using the correct voltage is important for optimal performance.

7.**Power Consumption:**

- Servo motors consume varying amounts of power based on load, speed, and torque requirements. Power is typically measured in watts or amps.

8. **Response Time:**

- The time it takes for a servo motor to respond to a control signal and reach the target position. This is important for time-sensitive applications like robotics and automation.

Servo Motor Control

Servo motor control is mainly done using **PWM (Pulse Width Modulation)** signals. The width of the pulse (usually between 1ms and 2ms) determines the motor's position. For example, a 1ms pulse moves the motor to 0 degrees, 1.5ms to 90 degrees, and 2ms to 180 degrees. The motor's internal control circuit uses feedback from a position sensor to adjust the motor's movement and reach the desired position. Microcontrollers like STM32 or Arduino generate PWM signals to control the servo's position. Servo motors are ideal for precise control in robotics, automation, and remote control systems.

Applications of Servo Motors

Servo motors are used in a variety of applications that require precise control of movement. Common uses include robotics, automated manufacturing, remote-controlled vehicles, drones, cameras, HVAC systems, medical equipment, and industrial automation. They are also found in home automation systems, like automatic doors and window blinds. Their ability to provide accurate positioning and smooth motion makes them ideal for these tasks.

Benefits of using servo motors

Servo motors offer high precision, accuracy, and fast response time, making them ideal for applications requiring precise movement. They provide high torque at low speeds, are energy-efficient, and compact in size. Their reliability and minimal maintenance make them suitable for long-term use in robotics, automation, and other precision systems.

2.5 JUMPER WIRES

Fig: Jumper wires

Jumper wires are essential tools in electronics for making temporary connections between different points in a circuit, often on breadboards or prototyping boards. They're widely used in IoT projects, robotics, and general electronics prototyping. Here's a detailed explanation of jumper wires, their types, uses, and applications:

1. **Purpose of Jumper Wires**

-Jumper wires are used to **connect components without soldering**, enabling easy, flexible, and temporary connections in electronic circuits.

2. **Types of Jumper Wires by Connector**

-Male-to-Male: Both ends have male connectors (pins), commonly used to connect two female header pins on a breadboard or development board.

-Female-to-Female: Both ends have female connectors, useful for connecting two male header pins, like those on modules or sensors.

-Male-to-Female: One end has a male connector and the other a female connector, ideal for bridging between male and female headers.

3. **Color Coding**

-Jumper wires come in various colors (e.g., red, black, yellow, green) to help differentiate between connections. For example, red often denotes power, black for ground, and other colors for signals, helping to keep the circuit organized.

4. **Common Lengths and Sizes**

Jumper wires are available in various lengths, from short (e.g., 5 cm) to longer sizes (e.g., 30 cm or more), to accommodate different circuit setups and board sizes.

5. **Materials**

Jumper wires are typically made of copper or aluminum with a PVC insulation cover, which makes them durable, flexible, and resistant to short circuits.

6. **Connection Compatibility**

-They are compatible with breadboards, Arduino, Raspberry Pi, ESP32, and most other prototyping boards, making them versatile for a wide range of projects.

7. **Use with Breadboards**

-Jumper wires are mainly used on breadboards, where they connect various components without the need for soldering, allowing rapid prototyping and testing of circuits.

8. **Flexible and Reusable**

-They can be disconnected and reconnected multiple times, making them ideal for experimental setups and temporary circuits. They're easy to rearrange when circuit adjustments are needed.

9. **Low Resistance**

-Jumper wires have low resistance, which ensures minimal voltage drop across the wire, making them reliable for most low-power applications.

10. **Insulation for Safety**

-The PVC or silicone insulation on jumper wires protects against accidental shorts and allows safe handling, even when current flows through them.

11. **Ideal for Rapid Prototyping**

-Jumper wires allow developers and engineers to quickly test, modify, and troubleshoot circuits without the need for soldering, making them perfect for prototyping.

12. **Easy to Use with Sensors and Modules**

-Many sensors and modules have male or female header pins, and jumper wires make it easy to connect them directly to a microcontroller or breadboard.

13. **Applications in IoT and Robotics**

-Jumper wires are extensively used in IoT, robotics, and DIY electronics projects for connecting sensors, actuators, LEDs, and other components to microcontrollers and development boards.

14. **Affordable and Widely Available**

-Jumper wires are inexpensive and readily available in various sizes and connector types, often sold in bulk or kits, making them accessible for hobbyists and professionals alike.

15. **Safety Precautions**

-Ensure that the jumper wires are rated for the voltage and current of the circuit, as using undersized wires for high power applications could result in overheating or damage.

2.6 BREAD BOARD

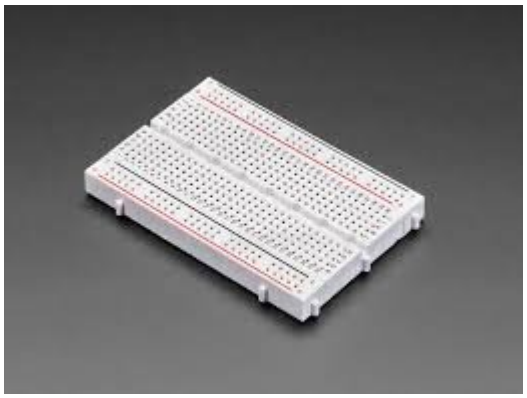


Fig: Breadboard

A breadboard is a widely used device for prototyping electronic circuits. Here are the key features of a breadboard:

1. Solderless Construction:

- Breadboards allow components to be connected without soldering, making it easy to modify or rearrange circuits.

2. Grid Layout:

- The board features a grid of holes in a standard 0.1-inch (2.54 mm) spacing, accommodating most electronic components and ICs.

3. Connection Strips:

- Terminal Strips:The central area contains rows of interconnected holes where components and wires can be inserted. Each row is electrically connected, allowing easy placement of components.
- Bus Strips:The sides of the breadboard typically feature long, interconnected columns called bus strips or power rails, which are used to distribute power and ground connections across the board.

4. Separation of Power Rails:

- Many breadboards have power rails divided into two sections to allow for different voltage levels or to split the power distribution across the board.

5. Labeling:

- Breadboards often have alphanumeric labeling (letters for rows and numbers for columns) tohelp identify and reference specific holes or connections.

6. Dual Power Rails:

- Some breadboards feature dual power rails on both sides, providing additional convenience for complex circuits requiring multiple voltage levels.

7. Binding Posts:

- Larger breadboards may include binding posts for easily connecting external power supplies.

8. Reusability:
- Components can be inserted and removed multiple times without damaging the board, making it ideal for iterative prototyping and learning.

9. Component Compatibility:
- Breadboards are compatible with a wide range of components, including resistors, capacitors, LEDs, transistors, and integrated circuits (ICs).

10. Modular Design:
- Breadboards can be connected side-by-side or end-to-end to expand the working area for larger projects.

11. Plastic Base:
- The base of the breadboard is typically made of plastic with a non-conductive surface, ensuring electrical isolation between different connections.

12. Spring Clips:
- The internal structure consists of metal spring clips that grip the leads of components and wires securely, providing reliable electrical connections.

13. Compact and Portable:
- Breadboards come in various sizes, from small, single-unit boards to larger, multiple-unit boards, making them portable and suitable for various project sizes.

14. Cost-Effective:
- Breadboards are relatively inexpensive and widely available, making them accessible forhobbyists, students, and professionals.

15. Versatility:
- Breadboards are used for designing and testing circuits in educational settings, DIY projects, prototyping, and temporary setups before finalizing designs on a printed circuit board (PCB). These features make breadboards an indispensable tool for anyone involved in electronics, from beginners learning the basics to experienced engineers developing new prototypes.

SOFTWARE :

2.7 ARDUINO IDE(software)

Fig: Arduino IDE

The Arduino IDE (Integrated Development Environment) is an open-source software platform used to write, compile, and upload code to Arduino-compatible microcontrollers. It provides a simple, user-friendly interface designed for beginners and advanced users alike, making it a popular tool for developing and prototyping embedded systems and IoT projects.

Features of Arduino IDE

1. Code Editor: The IDE includes a basic text editor with syntax highlighting, auto-indentation, and bracket matching to make code writing more efficient and readable. It supports a programming language based on C/C++ with Arduino-specific libraries, simplifying the control of hardware components like sensors, LEDs, motors, and displays.

2. Libraries: The Arduino IDE offers a wide range of pre-built libraries that allow users to easily add functionalities to their projects. Libraries are collections of code that simplify complex tasks, like controlling sensors or displays, communicating via Wi-Fi or Bluetooth, and even handling advanced protocols. Users can download additional libraries directly through the IDE's Library Manager.

3. Sketches: In Arduino terminology, a sketch refers to the code written for an Arduino project. The IDE manages sketches in files with a `.ino` extension, enabling users to organize and save code for different projects. Each sketch typically consists of two main functions:
setup(): Runs once when the microcontroller is powered on or reset. Used to initialize settings or configure input/output pins.
loop(): Runs repeatedly after `setup()` and contains the main code to be executed continuously.

4. Compiler and Debugger: The IDE includes a built-in compiler that translates high-level code into machine-readable instructions for the microcontroller. Although the IDE lacks a full debugging tool, it supports serial debugging through the Serial Monitor feature, allowing users to display and troubleshoot real-time data from the Arduino board.

5. Serial Monitor: The Serial Monitor provides a way to send and receive data between the computer and the Arduino board over USB. It's essential for debugging and monitoring values, enabling users to interact with the microcontroller and observe outputs from sensors or other components.

6. Board and Port Selection: The Arduino IDE allows users to select the specific type of Arduino board they are using (e.g., Arduino Uno, Mega, Nano) from a drop-down menu. It also automatically detects the USB port connected to the board, simplifying the setup process for uploading code.

7. Upload Function: After writing and verifying code, users can upload it directly to the connected Arduino board using the Upload button. The code is sent to the microcontroller, where it is executed based on the instructions provided in the sketch.

How to Use the Arduino IDE

1. Install the IDE: The Arduino IDE is available for Windows, macOS, and Linux and can be downloaded from the official Arduino website. It's also available as a web-based IDE, called Arduino Web Editor, which allows users to work online.

2. Connect the Board: Connect the Arduino board to the computer via USB. The IDE should automatically detect the board, allowing it to be selected in the Tools menu under Board and Port.

3. Write Code: Write the sketch using the editor. Begin with basic functions like setup() and loop(), and include additional libraries if needed for specific hardware components.

4. Verify and Compile: Press the Verify button to compile the code and check for syntax errors. Any errors are displayed in the output window, helping users troubleshoot issues before uploading.

5. Upload Code: Once verified, press the Upload button to load the compiled code onto the Arduino board. The code is stored in the microcontroller's memory and will start executing immediately after upload.

6. Monitor and Debug: Use the Serial Monitor to view data output from the board in real-time. This is especially useful for checking sensor readings, debugging, and receiving feedback from the board during execution.

Advantages of Arduino IDE:

- **Beginner-Friendly:** The interface is simple and easy to use, making it accessible to people with limited coding experience.
- **Cross-Platform Support:** Works on multiple operating systems, including Windows, macOS, and Linux.
- **Extensive Library Support:** Includes a vast array of libraries, allowing easy integration with hardware components.

- **Community and Support:** The Arduino platform has a large, active community that shares tutorials, projects, and resources, making it easy for new users to learn and find help.

Limitations of Arduino IDE

- Limited Debugging Capabilities: Lacks an advanced debugger, which can make it challenging to troubleshoot complex programs.
- Basic Code Editor: It doesn't include features found in more advanced IDEs, such as version control integration or code refactoring tools.
- Suitable for Small-Scale Projects: While it's great for prototyping and learning, the IDE may not be ideal for large, complex projects that require more robust coding environments.

Arduino IDE Alternatives

- PlatformIO: An advanced IDE supporting multiple microcontrollers, including Arduino. It provides features like debugging, unit testing, and integration with Visual Studio Code.
- Atmel Studio: Designed for Atmel microcontrollers (which include Arduino's AVR chips), offering more advanced debugging tools.
- Visual Studio Code with Arduino Extension: Adds Arduino support to VS Code, allowing users to write, compile, and upload sketches while leveraging the advanced features of VS Code.

The Arduino IDE is a versatile, user-friendly development platform perfect for both beginners and experienced developers looking to create Arduino-based projects. It offers essential tools for writing, compiling, and uploading code while also providing access to a wide range of libraries and resources. With its simplicity and strong community support, the Arduino IDE continues to be a key tool for developing embedded systems and IoT applications.

2.8 IDLE (USING PYTHON)

Fig:Python IDLE

In Python, **IDLE** (Integrated Development and Learning Environment) is a simple integrated development environment that comes pre-installed with Python. It allows you to write, test, and debug Python code in an easy-to-use interface. Here are some features and uses of IDLE:

Key Features of IDLE:

1. **Interactive Shell**: The Python shell (REPL - Read, Evaluate, Print, Loop) allows you to type and execute Python commands one line at a time, offering immediate feedback. It's ideal for testing code snippets or exploring libraries interactively.
2. **Editor Window**: IDLE provides an editor window where you can write and save Python scripts (.py files). It includes basic features like syntax highlighting, indentation support, and line numbering to assist in writing clean and structured code.
3. **Debugger**: It comes with a simple debugger that allows you to step through your code, set breakpoints, and examine variables at different points in execution.
4. **Cross-Platform**: IDLE works on multiple platforms, including Windows, macOS, and Linux, providing a consistent environment across these systems.
5. **Simple Interface**: The IDLE interface is minimalist, making it suitable for beginners. It's not as feature-rich as other IDEs like PyCharm or Visual Studio Code but is great for simple tasks and learning.

How to Use IDLE:

1. **Launching IDLE**: After installing Python, you can launch IDLE from the Start menu on Windows, or by typing `idle` in the terminal on macOS and Linux.

2. **Writing Code**: You can write code directly in the IDLE shell for quick experiments or use the editor window for more complex scripts.
3. **Running Code**: Once your script is written, you can execute it by pressing `F5` or selecting "Run" from the menu.
4. **Debugging**: You can set breakpoints by clicking next to the line numbers in the editor, and use the debugger to step through the code.

2.9 Ada fruit IO

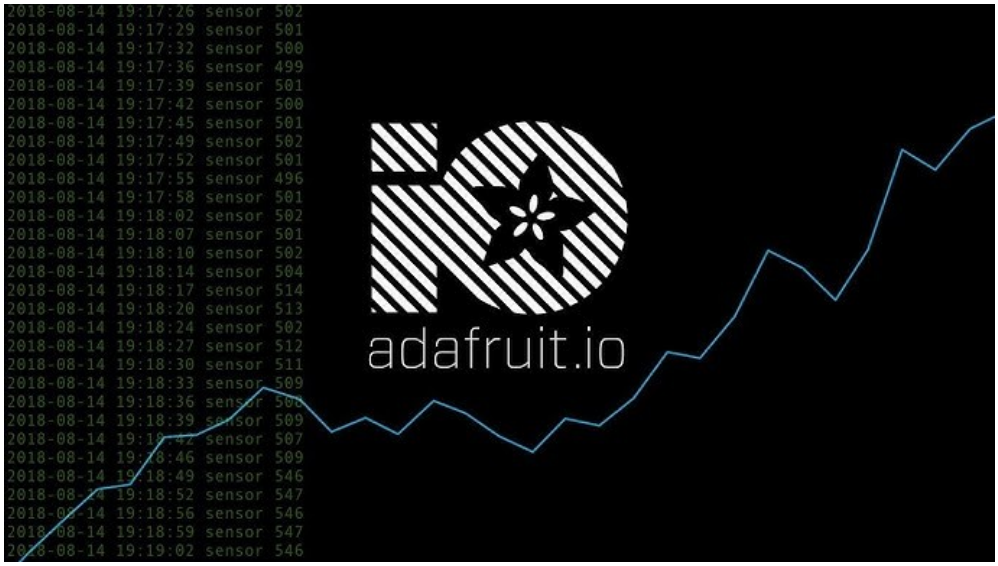


Fig:adafruit.io

Adafruit IO is a cloud-based service developed by Adafruit, designed to simplify the process of connecting your devices to the Internet of Things (IoT). It allows users to send, receive, and store data from sensors, actuators, or microcontrollers, and provides easy integration with various hardware platforms like Arduino, Raspberry Pi, and ESP8266/ESP32.

Key Features of Adafruit IO:

1. **Data Logging**:
   Adafruit IO stores data sent from your devices, allowing you to track and visualize it over time. This is useful for applications like temperature monitoring, humidity tracking, and other sensor data storage.
2. **Dashboards**:
   Users can create customizable dashboards to display real-time data in an intuitive and easy-

to-understand format. These dashboards can include graphs, gauges, and other widgets for visualizing data.

3. **Feeds**:
   Feeds in Adafruit IO are like data channels where devices send or receive data. Each feed can represent a specific data point, such as temperature, humidity, or the status of a sensor.
4. **Triggers and Actions**:
   You can set up triggers to automate actions based on incoming data. For example, you can set a trigger to turn on a light or send an email when a specific temperature threshold is reached.
5. **REST API**:
   Adafruit IO provides a REST API that allows you to interact with the platform programmatically. You can send and retrieve data from feeds, manage dashboards, and configure triggers via HTTP requests.
6. **MQTT Support**:
   Adafruit IO supports MQTT (Message Queuing Telemetry Transport), a lightweight messaging protocol commonly used in IoT. This allows for real-time, low-latency communication between your devices and the cloud.
7. **Security**:
   Adafruit IO offers secure access using authentication tokens, ensuring that only authorized users or devices can access your data or control your IoT devices.
8. **Free Tier and Pricing**:
   Adafruit IO offers a free plan with a limited number of feeds and data points. For more advanced use cases or higher limits, users can upgrade to paid plans.

Common Applications:

- **Home Automation**: Use Adafruit IO to control home appliances like lights, fans, or security cameras remotely.
- **Environmental Monitoring**: Track temperature, humidity, and other environmental factors with sensors connected to Adafruit IO.
- **Wearables and Health Monitoring**: Collect data from wearables or health devices, visualize it, and make real-time decisions based on the data.
- **Remote Monitoring and Control**: Monitor industrial machines or systems in real time and control them remotely.

Adafruit IO plays an important role in enabling remote monitoring and control.

1. **Data Monitoring**:
   Adafruit IO allows you to track and display real-time information about the door locking system, such as the status (locked or unlocked), RFID tag scans, and any access events. The data can be visualized on a custom dashboard with charts, gauges, or status indicators.
2. **Remote Control and Feedback**:
   Through Adafruit IO, you can remotely monitor and control the door lock. For example, if you want to unlock the door remotely, you can trigger an action via Adafruit IO, which sends a signal back to your STM32F411RE microcontroller to control the servo motor.
3. **Event Logging**:
   Adafruit IO logs events like successful or failed RFID scans. This data is stored on the platform and can be accessed later for review, allowing for event tracking, user history, and even security audits.
4. **IoT Integration**:
   With Adafruit IO's MQTT or REST API support, you can easily integrate the system with other IoT devices or platforms. It simplifies the process of connecting your STM32 microcontroller to the internet, removing the need for a complex Wi-Fi module setup.
5. **Automation**:
   You can set up triggers on Adafruit IO, such as sending a notification when a certain tag is scanned or when the door is left open for too long. This adds an extra layer of security and automation to your system.
6. **Data Analytics**:
   The platform allows you to analyze trends over time, such as which users are accessing the system most frequently or if there are any unusual access patterns. This helps improve the security and efficiency of the system.

CHAPTER-3

3.1 CIRCUIT DIAGRAM

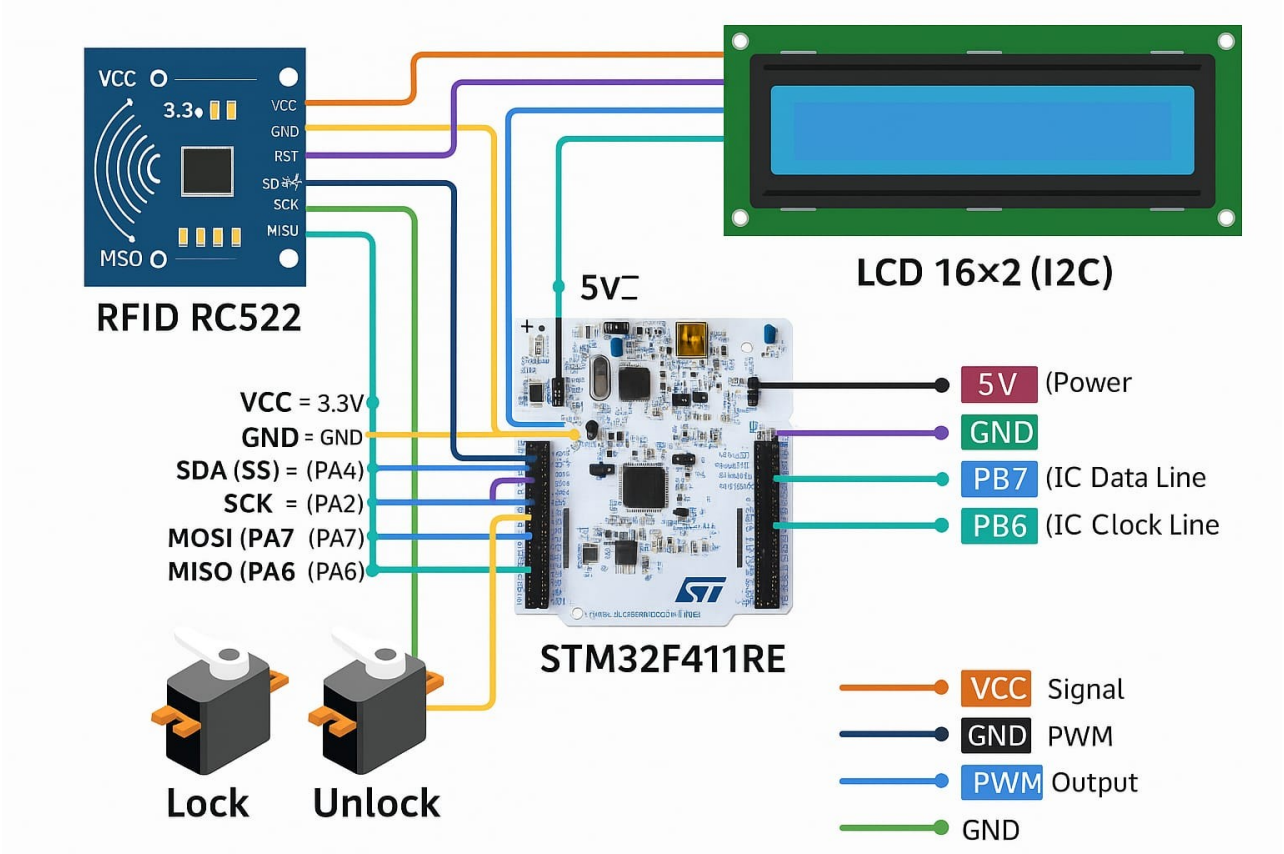


Fig: Circuit diagram

3.2WORKING PROCESS

1. **RFID Scanning**:
   - The system starts when an RFID tag is scanned by the **RC522 RFID reader**.
   - The **STM32F411RE** microcontroller processes the data from the RFID tag, identifying the unique tag ID.

2. Authentication:

- The microcontroller compares the scanned RFID tag ID with a predefined list of authorized users stored in memory (e.g., in EEPROM or a database).
- If the tag is **authorized**, the system proceeds to unlock the door. If not, it remains locked and can send a failure message or log the event.

3. Controlling the Servo Motors:

- Upon successful authentication, the STM32F411RE sends a signal to the **servo motors** (connected to pins PB10 and PB4).
- The servo motors control the **mechanical locking mechanism** to either **unlock or lock** the door, depending on the desired action.

4. Displaying Information on LCD:

- The **16x2 I2C LCD display** provides real-time feedback to the user. For example:
  - "Door Unlocked" when authentication is successful.
  - "Access Denied" if the tag is unauthorized.
- The display also shows other relevant system statuses (e.g., RFID scanning, motor movement).

5. Data Logging and Remote Monitoring via Adafruit IO:

- Data such as successful and failed RFID scans, door status (locked/unlocked), and user access logs are sent from the STM32F411RE to **Adafruit IO** via **Python code** running on a connected PC or microcontroller.
- **Adafruit IO** stores this data and displays it on a custom dashboard. The dashboard shows real-time events and status updates, such as:
  - Access logs and timestamps.
  - System status (locked or unlocked).

6. Remote Control and Notifications:

- Through **Adafruit IO**, users can remotely control the system by sending commands (e.g., unlock the door via the dashboard).
- **Triggers** can be set to send notifications (e.g., an email or a text message) when certain events occur, like unauthorized access attempts.

7. Security and Automation:

- If there are any security-related events (e.g., failed access attempts or unusual activity), Adafruit IO can trigger alerts or notifications.
- You can set **automation rules** on Adafruit IO to perform actions based on data changes, like sending an alert when a specific RFID tag is scanned or if the door is left open for a long period.

8. **Continuous Monitoring**:

- The system continuously monitors the status of the door lock, scanning RFID tags, and logging data to Adafruit IO.
- The system remains connected to Adafruit IO, ensuring that all data is synchronized in real time.

Arduino IDE Working Process:

The Arduino IDE provides a seamless workflow for embedded programming, from writing and compiling code to uploading and debugging it on a microcontroller. By following this process, users can quickly develop and deploy Arduino-based projects, making it a popular tool for learning, prototyping, and implementing IoT and embedded systems projects.

The working process of an advanced IoT-based current surge safeguard system for electric vehicles (EVs) involves monitoring, detecting, and mitigating current surges to protect the vehicle's electrical components. This process combines sensor data, real-time processing, and IoT connectivity to enhance safety and reliability. Here's an outline of how such a system operates:

1. System Initialization:

- Microcontroller and Sensor Setup: The microcontroller (e.g., Arduino or ESP32) initializes, setting up all necessary configurations for connected sensors, such as current sensors, temperature sensors, and voltage monitors.
- IoT Module Configuration: The IoT communication module (e.g., Wi-Fi or GSM) establishes a connection with the network, enabling real-time data transmission to cloud servers or a monitoring application.

2. Real-Time Monitoring:

- Current Measurement: Current sensors continuously monitor the current flowing through critical EV components like the battery, motor, and control circuits.
- Temperature and Voltage Monitoring: Additional sensors monitor other parameters, such as temperature and voltage, which are also affected by surges and may signal potential issues.
- Data Logging: The microcontroller collects and logs data from these sensors, checking periodically for any values that exceed preset safety thresholds.

3. Surge Detection:

- Threshold Analysis: The system compares real-time sensor values with predefined safe limits stored in the microcontroller. If the current exceeds the threshold, it indicates a potential surge.
- Pattern Recognition (Advanced Models): Advanced versions of this system can use machine learning algorithms to identify patterns or anomalies in current flows, potentially detecting issues before they exceed thresholds.

4. Surge Mitigation:

- Relay Activation: In case of a detected surge, the microcontroller activates a relay module, which can disconnect the power supply from the affected components to prevent damage.
- Cooling Mechanism Activation: If the system detects a temperature rise associated with the surge, it can trigger cooling mechanisms (e.g., fans or thermal cutoffs) to dissipate excess heat.
- Alert System: The system may also trigger an audible alarm or LED indicator to notify the driver of the issue.

5. IoT-Based Alerts and Data Transmission:

- Cloud Integration: The microcontroller, through the IoT module, sends real-time data to a cloud platform or monitoring dashboard, allowing remote supervision and data logging for future analysis.
- User Notification: In the event of a surge or any abnormal condition, notifications are sent to the user via a connected smartphone app, SMS, or email. These alerts help the user respond quickly and minimize risk.

6. System Recovery and Data Analysis:

- System Reset: Once the surge has subsided or the abnormal condition is resolved, the microcontroller resets the system to resume normal operation.
- Data Analysis and Reporting: The stored data, including historical surge events, is analyzed periodically. This analysis can help improve system settings, refine threshold values, and even provide predictive maintenance insights.

7. Continuous Learning and Updates:

- Machine Learning for Pattern Recognition: If machine learning is implemented, the system periodically updates its algorithms to improve detection accuracy based on new data.
- System Updates: Software updates for the microcontroller and IoT modules can be applied over the air (OTA) if needed, enhancing security and functionality.

3.3 Code

```
#include <Wire.h>
#include <LiquidCrystal_PCF8574.h>
#include <SPI.h>
#include <MFRC522.h>
#include <Servo.h>

// Define pin connections
#define I2C_ADDR 0x27  // LCD I2C Address
#define SS_PIN PA4     // RFID Slave Select (SDA on RC522)
#define RST_PIN PB0    // RFID Reset pin for RC522
#define RELAY_PIN PA8  // Relay pin for access control
#define SERVO1_PIN D9  // Servo 1 control pin
#define SERVO2_PIN D6  // Servo 2 control pin
#define SDA_PIN PA7    // SPI MOSI (connects to RFID module MOSI)
#define SCK_PIN PA5    // SPI Clock (connects to RFID module SCK)
#define MISO_PIN PA6   // SPI MISO (connects to RFID module MISO)
#define MOSI_PIN PA7   // SPI MOSI (connects to RFID module MOSI)

// Initialize components
LiquidCrystal_PCF8574 lcd(I2C_ADDR);
MFRC522 mfrc522(SS_PIN, RST_PIN);
Servo servo1;
Servo servo2;

#define ACCESS_DELAY 2000  // 2 seconds for access granted
#define DENIED_DELAY 1000  // 1 second for access denied

String stm32 = "";

void setup() {
  // Initialize I2C LCD
  Wire.begin();
  lcd.begin(16, 2);
  lcd.setBacklight(255);
  lcd.setCursor(0, 0);
  lcd.print("Hello, Nucleo!");
  lcd.setCursor(0, 1);
  lcd.print("I2C LCD Ready");
```

```
  // Initialize serial communication
  Serial.begin(9600);
  SPI.begin();
  mfrc522.PCD_Init();

  // Initialize relay and servo pins
  pinMode(RELAY_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, LOW);

  servo1.attach(SERVO1_PIN);
  servo2.attach(SERVO2_PIN);
  servo1.write(0);
  servo2.write(180);
  delay(ACCESS_DELAY);
  // Serial.println("Place your RFID card on the reader...");
}

void loop() {
  if (!mfrc522.PICC_IsNewCardPresent()) return;
  if (!mfrc522.PICC_ReadCardSerial()) return;

  // Serial.print("Card UID: ");?
  String content = "";
  for (byte i = 0; i < mfrc522.uid.size; i++) {
    content.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
    content.concat(String(mfrc522.uid.uidByte[i], HEX));
  }
  content.toUpperCase();
  // Serial.println(content);

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Card Detected!");
  lcd.setCursor(0, 1);
  lcd.print(content);

   if (content.substring(1) == "C3 9B 2C DA" || content.substring(1) == "5A AD 31
02") {
```

```
    // Serial.println("Access Granted!");
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Access Granted!");
    digitalWrite(RELAY_PIN, HIGH);

    servo1.write(0);
    servo2.write(180);
    stm32 = "/authorized/" + content + "/";
    Serial.println(stm32);

    delay(ACCESS_DELAY);

    servo1.write(90);
    servo2.write(90);
    delay(ACCESS_DELAY);

    servo1.write(0);
    servo2.write(180);
    delay(ACCESS_DELAY);
    digitalWrite(RELAY_PIN, LOW);
  } else {
    // Serial.println("Access Denied");
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Access Denied!");
    stm32 = "/UN - authorized/" + content + "/";
    Serial.println(stm32);

    delay(DENIED_DELAY);
  }
}
```

3.3.1 Python Code :

```
import serial
import time
from Adafruit_IO import Client, Feed, RequestError

# Adafruit IO setup
ADAFRUIT_IO_KEY = 'aio_tbRS14QVYyxwpsXNrmqm29lgRJ7g'  # Replace with your Adafruit IO key
ADAFRUIT_IO_USERNAME = 'ramisetty_pavani'  # Replace with your Adafruit IO username
aio = Client(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

# Feeds setup (Ensure these feeds exist in your Adafruit IO dashboard)
try:
    status_feed = aio.feeds('status')  # Replace 'status' with your feed name for status
     number_feed = aio.feeds('number')  # Replace 'number' with your feed name for number
except RequestError:  # Create the feeds if they don't exist
    status_feed = aio.create_feed(Feed(name='status'))
    number_feed = aio.create_feed(Feed(name='number'))

# Set up the serial port connection
ser = serial.Serial(
      port='/dev/ttyACM0',      # Replace with your port (e.g., 'COM3' for Windows, '/dev/ttyS0' for Linux)
    baudrate=9600,        # Set baudrate (ensure this matches your device)
    timeout=1             # Timeout in seconds
)

# Function to read data from the serial port and extract values between slashes
def read_serial_data():
    try:
        if ser.is_open:
            data = ser.readline().decode('utf-8').strip()
            if data:
                print(f"Received: {data}")

                # Split the data based on the slash '/'
                parts = data.split('/')
```

```
            if len(parts) >= 3:
                status = parts[1].strip()  # Data between 1st and 2nd slash (status)
                number = parts[2].strip()  # Data between 2nd and 3rd slash (number)

                print(f"Status: {status}")
                print(f"Number: {number}")

                # Send data to Adafruit IO
                send_to_adafruit_io(status, number)

                return status, number
            else:
                print("Data format is incorrect. Expected two slashes.")
    except serial.SerialException as e:
        print(f"Error reading from serial port: {e}")
    return None

# Function to send data to Adafruit IO
def send_to_adafruit_io(status, number):
    try:
        # Send the status and number to their respective feeds
        aio.send_data(status_feed.key, status)
        aio.send_data(number_feed.key, number)
        print(f"Status and Number sent to Adafruit IO.")
    except Exception as e:
        print(f"Error sending data to Adafruit IO: {e}")

if __name__ == "__main__":
    print("Starting serial port reading and Adafruit IO update...")
    try:
        while True:
            serial_data = read_serial_data()
            if serial_data:
                # Further processing of the separated variables can be done here
                pass
            time.sleep(1)
    except KeyboardInterrupt:
        print("Stopping serial port reading.")
    finally:
        ser.close()
```

CHAPTER-4

4.1 OUTPUT

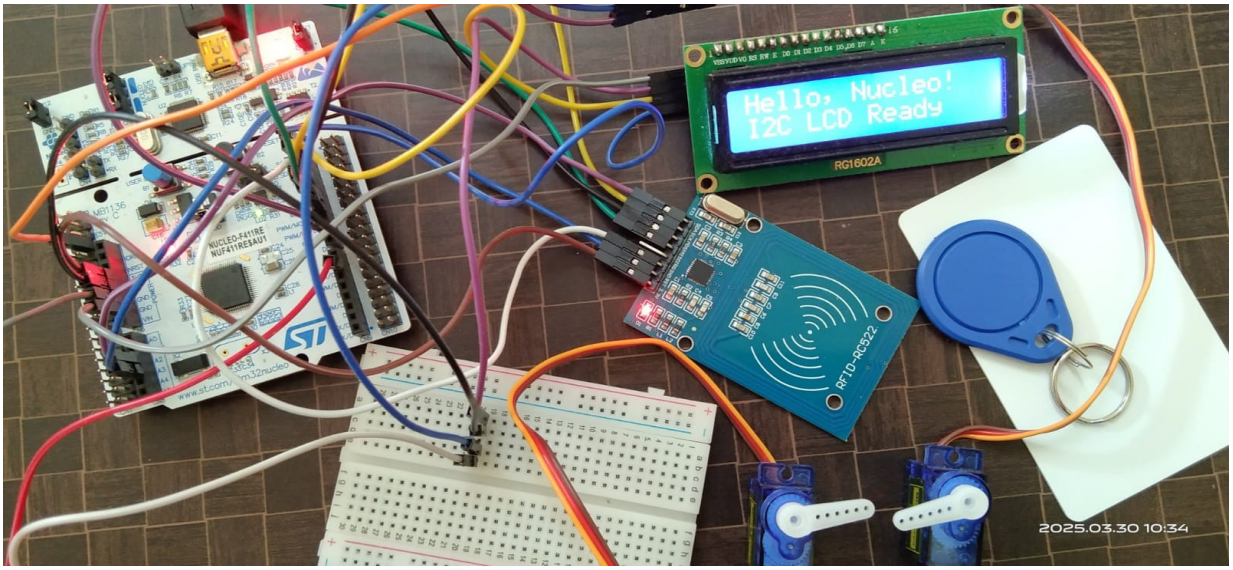
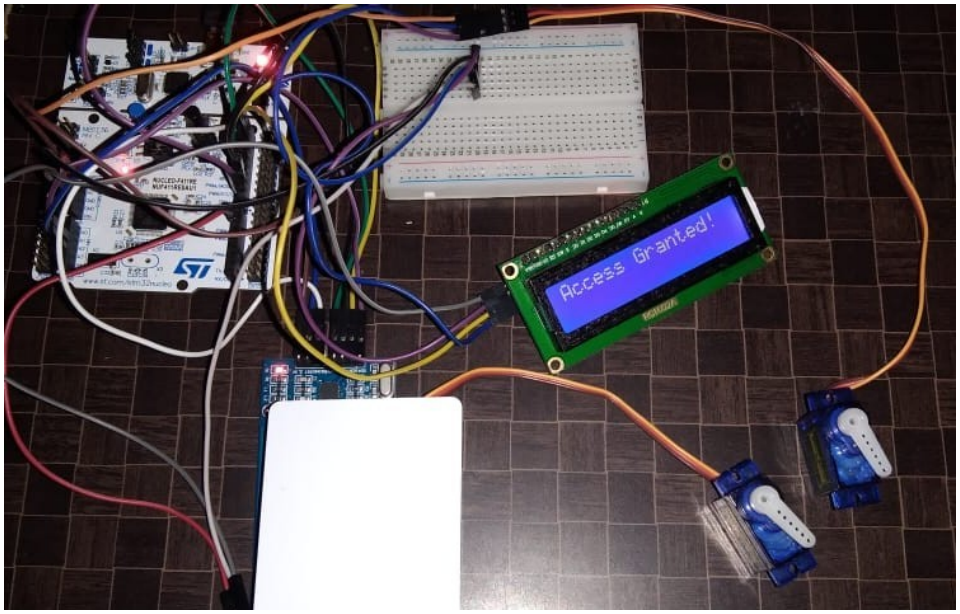
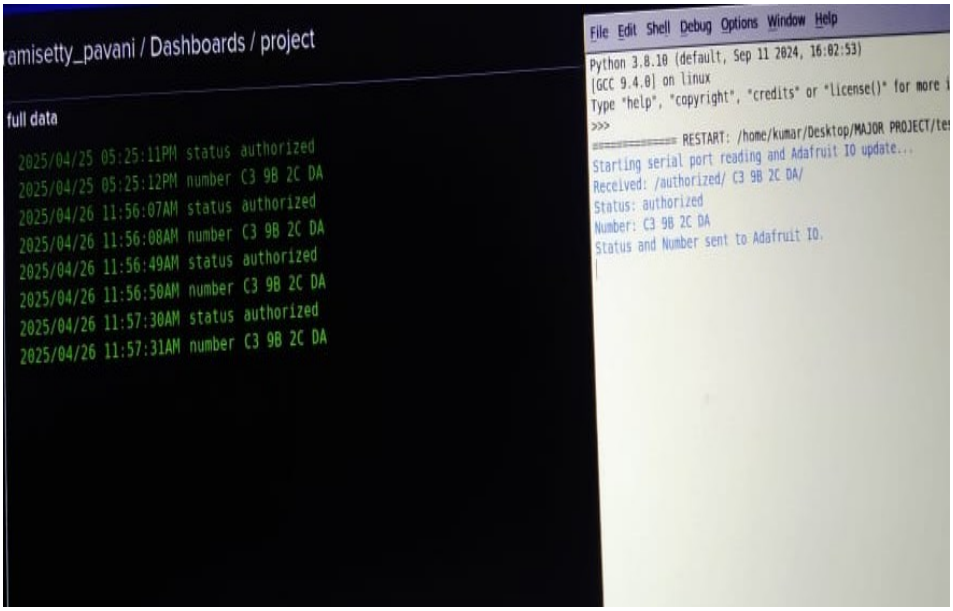


Fig: Output

ACCESS GRANTED :

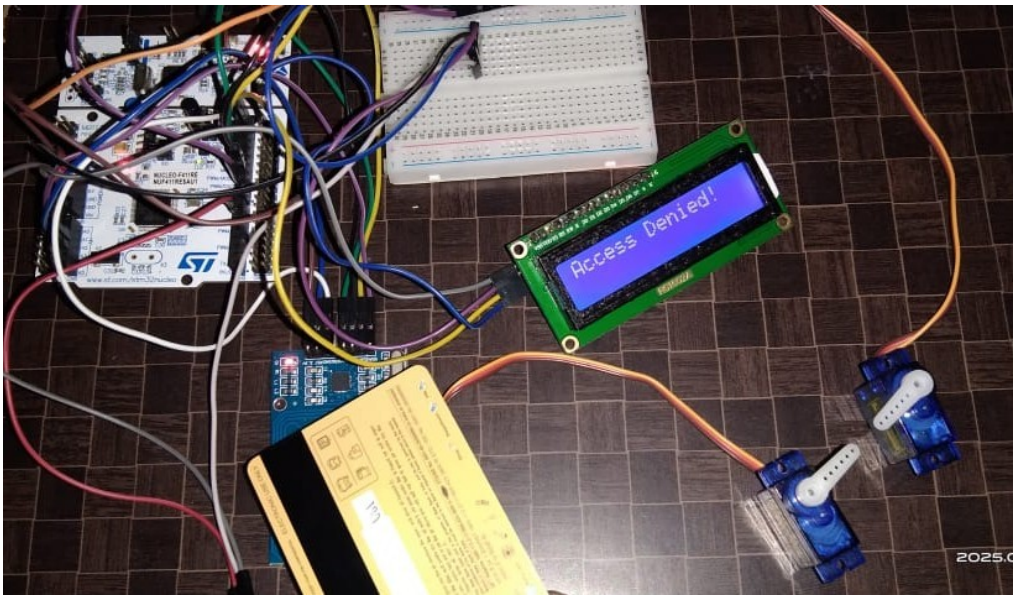


F*ig: Access Granted*

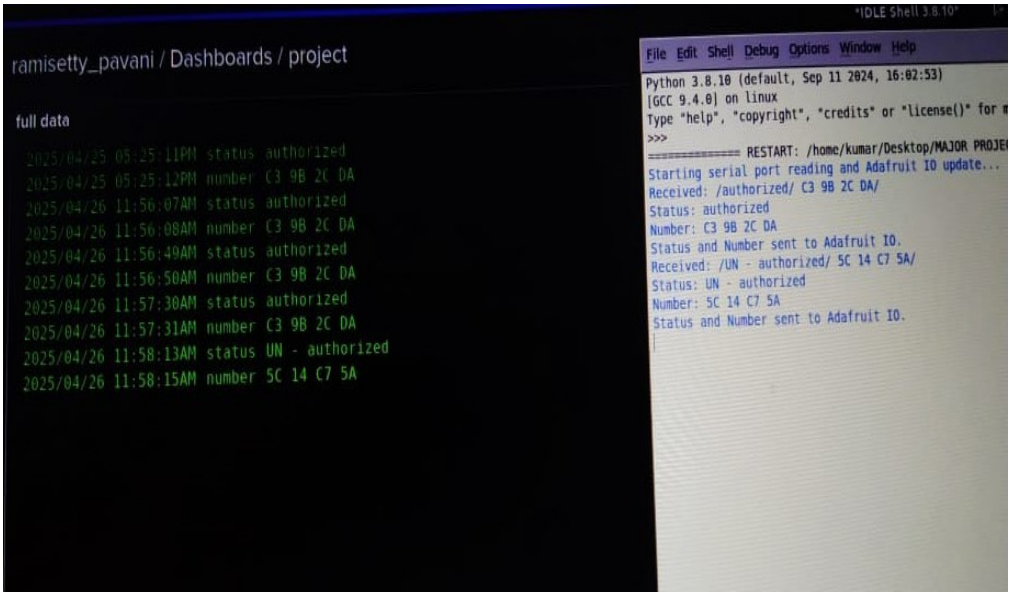


F*ig: Access Granted Out Put in Ada Fruit IO*

ACCESS DENIED :



F*ig: Access Denied*



F*ig: Access Denied Out Put in Ada Fruit IO*

CHAPTER-5

5.1 ADVANTAGES

1. Enhanced Security:

- **RFID authentication** provides secure and contactless access control, reducing the risk of key duplication or loss.
- The system can store and verify multiple RFID tags, making it flexible for different users.

2. Remote Monitoring:

- Using **Adafruit IO**, the system allows **real-time monitoring** of the door status (locked/unlocked) and access events from anywhere via a web dashboard.
- **Data logging** in Adafruit IO helps track user access and activity over time for security audits.

3. Ease of Use:

- The system provides clear feedback on the **LCD display**, indicating whether access was granted or denied.
- The **Adafruit IO dashboard** makes it easy to monitor and control the system remotely without complex setups.

4. Scalability and Flexibility:

- The system can easily be **expanded** to include more RFID tags, users, or even additional locks.
- The integration with **Adafruit IO** allows the system to be scaled for various applications, such as home automation or office security.

5. Automation and Alerts:

- With **triggers and automation** in Adafruit IO, the system can send alerts or perform actions automatically, such as sending notifications when unauthorized access is detected or when the door is left open.

6. Cost-Effective:

- Using an **STM32F411RE** microcontroller and affordable **RFID components** (like the RC522 reader), the system is a **low-cost solution** compared to other high-end access control systems.

7. User-Friendly Interface:

- The **Adafruit IO platform** offers a simple, intuitive interface for visualizing and controlling system data.
- The **LCD display** provides instant, local feedback, making the system more user-friendly.

8. Customizable and Programmable:

- The system is highly **customizable** through the microcontroller's programming, allowing changes in functionality, features, and behavior based on user needs.
- The Python programming in Adafruit IO allows for further integration and additional IoT functionalities.

9. Power Efficiency:

- The system can be designed to be **power-efficient**, especially when using low-power components like the STM32F411RE, making it suitable for long-term, continuous operation.

5.2 DISADVANTAGES

1. Limited Range of RFID Tags:

- RFID systems typically have a limited read range, which may require the user to position the tag very close to the reader, potentially causing slight delays or inconvenience in high-traffic areas.

2. Dependency on External Services:

- Although basic functionality does not require internet access, the **Adafruit IO** platform requires an internet connection for remote monitoring and control. If the internet is down, access to the dashboard or data logging features would be unavailable.

3. Potential for System Overload:

- If the system is handling a large number of RFID tags or frequent access events, **network latency** or system performance issues could arise, especially when the system needs to send large amounts of data to **Adafruit IO**.

4. Complexity in Setup:

- Setting up the system, especially integrating the **STM32F411RE**, **RC522 RFID**, **servo motors**, and **Adafruit IO**, can be complex for beginners without prior experience in embedded systems and IoT development.
- Proper wiring, programming, and configuration of both the hardware and software components may require detailed knowledge.

5. Power Consumption:

- While the system can be designed for power efficiency, the **servo motors** and other components may require significant power, especially for continuous operation or when handling multiple access events. This could lead to higher power consumption, especially if the system is used extensively.

6. Limited Physical Security:

- While the RFID tag-based system provides digital security, it does not address physical security concerns, such as **tampering** with the reader or the lock itself. A determined intruder could potentially bypass the system by physically accessing components.

7. Hardware Failure Risks:

- Like any hardware-based system, the **RFID reader**, **servo motors**, or the **microcontroller** could malfunction or get damaged. In such cases, the system would stop functioning correctly, requiring repairs or replacements.

8. Cost of Advanced Features:

- While the basic system is affordable, adding more advanced features like **encryption for RFID communication**, **cloud storage**, or **multiple RFID scanners** may increase the overall cost of the system.

9. Limited Integration with Other Security Systems:

- The system may not integrate easily with other **advanced security systems** like biometric or facial recognition systems, limiting its flexibility if you want to upgrade to a more comprehensive security solution in the future.

5.3 APPLICATIONS:

1. **Home Security Systems**:

- The system can be used in homes to control access to doors, gates, or other secure areas. Homeowners can set up RFID tags for family members or trusted individuals, ensuring a secure and contactless entry.

2. Office or Commercial Building Access:

- For businesses, this system provides an easy way to manage employee access to restricted areas. RFID tags can be issued to employees, granting access to specific rooms or floors within a building.

3. Smart Homes and Automation:

- The system can be integrated with other smart home devices like lights, alarms, and temperature control, enhancing home automation. For example, when the door unlocks, the system could trigger a smart home device to adjust the lighting or air conditioning.

4. Hotel Room Access:

- In hotels, RFID-based door locks can be used for room access, providing a seamless check-in experience for guests. The system can automatically log guest entry and exit times and can be managed remotely.

5. Schools and Universities:

- Schools and universities can use RFID-based door locks for access control in dormitories, classrooms, laboratories, and administrative offices. The system provides an efficient way to monitor and control access, ensuring security and privacy for students and staff.

6. Libraries and Restricted Areas:

- Libraries or research facilities can use RFID to control access to restricted areas or special collections. It can also track access events for auditing purposes.

7. Warehouses and Industrial Facilities:

- In warehouses, factories, and industrial facilities, RFID-based access control ensures that only authorized personnel can access sensitive areas, such as inventory rooms or production floors.

8. Fitness Centers:

- Gyms and fitness centers can use the system to grant members access to the facility or specific areas like lockers or restricted training zones. The system can also log entry and exit times for security purposes.

9. Hospital and Healthcare Facilities:

- RFID-based locks can be used in healthcare environments to control access to medication rooms, patient wards, or restricted areas like operating rooms, enhancing security and preventing unauthorized access.

10. Public or Private Parking Lots:

- The system can be used in parking garages or private parking lots for vehicle entry and exit. RFID tags attached to vehicles allow automatic opening of gates without the need for physical keys.

11. IoT and Remote Monitoring:

- With Adafruit IO integration, this system can be part of a larger IoT ecosystem for remote monitoring of access events, door status, and other related data. This makes it ideal for managing multiple door systems remotely.

12. Military and Government Facilities:

- For secure government buildings, military bases, or sensitive operations, RFID-based door locking systems ensure that only authorized personnel can access restricted zones, with full event tracking and auditing for security purposes.

CHAPTER-6

6.1 CONCLUSION

In conclusion, the **RFID-based door locking system** using the **STM32F411RE** microcontroller and **Adafruit IO** provides a modern, secure, and efficient solution for access control. By leveraging the power of RFID technology, this system ensures contactless authentication, eliminating the risks associated with traditional key-based systems. The integration of **servo motors** for locking/unlocking and the **I2C LCD** for real-time feedback makes the system user-friendly and reliable.

The addition of **Adafruit IO** for cloud-based data logging and remote monitoring enhances the system's capabilities, providing users with the ability to track access events, automate actions, and receive notifications, all while simplifying the setup process. This makes the system not only secure but also adaptable to various use cases, such as in homes, offices, schools, or industrial applications.

While there are some limitations, such as the range of RFID tags and potential security vulnerabilities, these can be mitigated with proper encryption and system design. Overall, this project demonstrates the power of combining **embedded systems**, **IoT**, and **cloud technologies** to create a versatile and scalable solution for modern access control needs.

6.2 REFERENCES

- ***STM32F411RE Microcontroller – www.st.com***
- **RFID RC522 Module** – MFRC522 Datasheet
- **RFID Library for Arduino** – https://github.com/miguelbalboa/rfid
- **LCD 16x2 with I2C Display** – DFRobot LCD Module
- **Servo Motor (SG90/MG90S)** – SG90 Datasheet
- **Adafruit IO Platform** – https://io.adafruit.com
- **Adafruit IO Python Library** – https://github.com/adafruit/Adafruit_IO_Python
- **Python IDLE Documentation** – https://docs.python.org/3/library/idle.html