ABSTRACT

EEPROM with Arduino states the importance of empowering Arduino-based embedded systems to handle data integrity and to manage intense tasks aiming for data security in much broader applications like industrial automation. Our project focuses on overcoming the issue of limited data storage, data corruption and random data access. Here we use an External EEPROM to achieve the desired outcome. The project EEPROM with Arduino results in enhanced data storage, fortified data security and furthermore data integration. This project offers a practical and user-friendly way to store data in EEPROM memory, enabling users to collect and save information from sensors, input devices, or other sources. The Arduino platform provides a simple and flexible interface for programming and interfacing with the EEPROM, making it accessible to both beginners and experienced developers. The system is designed to log and store data efficiently. Users can easily adapt the project for various applications, such as environmental monitoring, IoT devices, and sensor data acquisition. This seamless integration simplifies data management and ensures that essential calibration data and user configurations are retained, even when the power is disconnected, enhancing the overall versatility and reliability of our project.

1. INTRODUCTION

In the world of embedded systems and microcontroller-based projects, data storage often becomes a pivotal challenge. Arduino, with its widespread popularity, offers an array of microcontrollers equipped with onboard Electrically Erasable Programmable Read-Only Memory (EEPROM) to cater to data storage needs. However, there are instances when the storage capacity of onboard EEPROM falls short, or data security becomes a paramount concern. To overcome these limitations, an exciting avenue emerges: the integration of external EEPROM modules with Arduino. This introduction sets the stage for an Arduino-based project that explores the integration of external EEPROM as a means to expand data storage capacity and enhance data security. This project venture marks an important evolution in the Arduino ecosystem, offering a powerful combination of versatility, scalability, and enhanced data management capabilities. In this exploration of external EEPROM using Arduino, we delve into the intricacies of interfacing, data management, and security considerations, demonstrating how this integration opens doors to a wide range of exciting and data-rich applications.

2. PROBLEM STATEMENT

In the realm of Arduino-based embedded systems, there is a persistent challenge related to data storage and security. Arduino microcontrollers come equipped with onboard Electrically Erasable Programmable Read-Only Memory (EEPROM), which serves as non-volatile memory for data storage. However, this onboard EEPROM has inherent limitations, such as limited storage capacity and potential vulnerability to data corruption or unauthorized access. The problem statement addressed by this project is the need for a robust and scalable solution to overcome these limitations by integrating external EEPROM modules with Arduino microcontrollers. The project aims to address these challenges by integrating external EEPROM modules with Arduino, creating a versatile, secure, and scalable data storage solution. The project seeks to enhance data storage capacity, fortify data security, provide scalability options, ensure data integrity, and deliver an intuitive user interface for interacting with stored data. By doing so, it aims to empower Arduino-based embedded systems to handle data-intensive tasks, manage data securely, and cater to a broader range of applications, from data logging to industrial automation and beyond. Hence our project aims to overcome the limitations of volatile memory by implementing a solution that leverages the capabilities of EEPROM, which allows data to be written and read without losing it when the device is turned off. The key challenges that we have addressed include designing an Arduino-based system that can write and read data to and from EEPROM, ensuring data integrity through error-checking mechanisms, optimizing the use of EEPROM to extend its lifespan, and potentially creating a user-friendly interface for interacting with the stored data. This project aims to provide a practical solution for applications where non-volatile data storage is critical, such as configuration settings, sensor data logging, or user-specific preferences.

3. PROJECT OBJECTIVES

The objective of our project involving EEPROM (Electrically Erasable Programmable Read-Only Memory) using Arduino is to create a data storage and retrieval system. The project aims to utilize the potentiometer as an intuitive input device, allowing users to interact with the system smoothly. EEPROM is a non-volatile memory that can store data even when the power is turned off. In this project, the Arduino microcontroller is utilized to write, read, and manage data in the EEPROM. The project may involve designing a user interface to input and retrieve data, implementing error-checking mechanisms to ensure data integrity, and optimizing the use of EEPROM to prolong its lifespan. The primary goal is to demonstrate the capability of Arduino in managing EEPROM memory for data storage and retrieval. By leveraging the ADC, we can obtain precise and accurate digital representations of the analog readings, enabling further processing and control within the Arduino. Storing this data in the EEPROM ensures data persistence, allowing critical settings and preferences to be retained even after power cycles. With a focus on simplicity and modularity, the project will provide an abstraction layer, streamlining development and encouraging broader application of this analog-to-digital conversion and EEPROM usage in Arduino projects. In this project, the Arduino microcontroller is utilized to write, read, and manage data in the EEPROM. This can have various applications, such as storing user settings, sensor readings, or configuration data that needs to be retained between power cycles. The project may involve designing a user interface to input and retrieve data, implementing error-checking mechanisms to ensure data integrity, and optimizing the use of EEPROM to prolong its lifespan. Overall, the primary goal is to demonstrate the capability of Arduino in managing EEPROM memory for data storage and retrieval.

4. METHODOLOGY

STEP 1: Connect the components as per the circuit diagram.

STEP 2: Using Arduino IDE implement the entire code. Verify the code completely and compile it. Once compiled, it's time to upload the code to its Arduino board.

STEP 3: After uploading the code open the serial monitor and set the **Baud rate as 9600** and select **Auto-scroll.**

STEP 4: Now the output will be displayed for each adjustment given in the potentiometer.

4.1. COMPONENTS

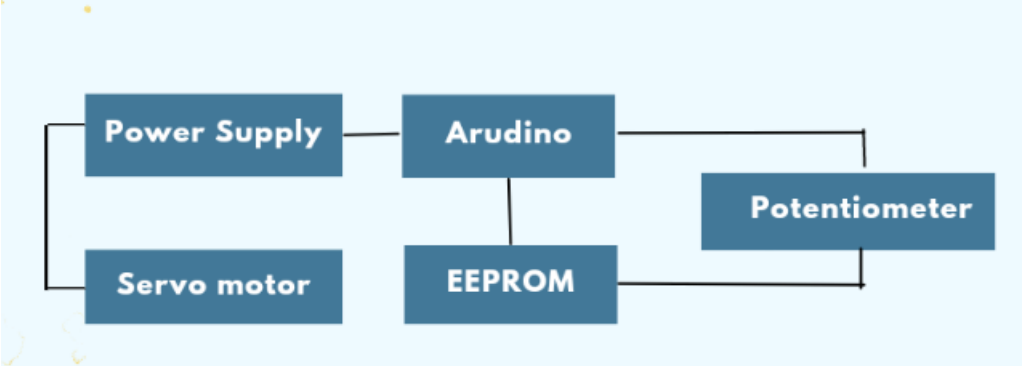
AT24LC256 EEPROM Module: The AT24LC256 EEPROM module can be used to store data that needs to be retained even when the power is turned off, such as configuration settings or user data.
You can interface the EEPROM module with a microcontroller or other digital devices through I2C or SPI communication protocols.
Write and read operations can be performed on the EEPROM to store and retrieve data as needed for your application.

Potentiometer (10K Linear Taper Potentiometer): A potentiometer can be used as an input device to set or adjust parameters related to the operation of the EEPROM or other components in your project.
For instance, you can connect the potentiometer to an analogue input pin on a microcontroller. The potentiometer's knob can be used to select memory addresses or set values like write delays, thresholds, or other parameters.

Servo Motor (SG90): A servo motor can be employed to physically control or interact with elements in your project.

You can use the servo motor to create a mechanism that writes data to or reads data from the EEPROM module. For example, the servo arm can press buttons or switches to trigger specific EEPROM operation

4.2. BLOCK DIAGRAM



5. OBSERVATION DONE

Observation done for our project involving external EEPROM using Arduino can add versatility, functionality, and efficiency to the system. Here are some ideas for potential future improvements; Wireless Connectivity: Integrate wireless communication modules (e.g., Wi-Fi, Bluetooth, LoRa) to enable remote access and data retrieval from the external EEPROM, enhancing the project's accessibility and remote monitoring capabilities. Cloud Integration: Develop a mechanism to synchronize data stored on the external EEPROM with cloud storage services (e.g., AWS, Google Cloud, Azure) for seamless data backup, analysis, and remote access. Real-Time Clock (RTC): Incorporate an RTC module to timestamp data entries, allowing for precise time-based data analysis and event logging. Battery Optimization: Implement power-saving techniques, such as sleep modes and wake-up triggers, to minimize power consumption in battery-powered applications. Expandable Storage: Develop a modular system that allows users to

add or swap external EEPROM modules easily, providing flexibility for different storage needs.

6. COMPLETE ANALYSIS OF PROJECT DONE

Write Data: The writeEEPROM function writes a byte of data to a specific memory address on the EEPROM. Read Data: The readEEPROM function reads a byte of data from a specific memory address on the EEPROM. The loop in the Arduino code writes data to the EEPROM and then reads it back. You can use this as a basis for various analyses. For example, you can store sensor data, timestamps, or configuration settings on the EEPROM and retrieve and analyze them later. To perform more advanced analyses, you may need to implement specific algorithms and data structures based on the data you are storing on the EEPROM.

7. TECHNOLOGY USED

Using an external EEPROM (Electrically Erasable Programmable Read-Only Memory) with an Arduino involves several components and technologies. EEPROMs are non-volatile memory devices that allow you to store data that persists even when the power is turned off. Here are the key technologies and components typically used in our project, Arduino is an open-source electronics platform based on easy-to-use hardware and software. External EEPROM is the main component of your project. You can use EEPROM chips such as the AT24C series (e.g., AT24C256, AT24C512) or other I2C/SPI-compatible EEPROMs. These chips have various capacities and communication interfaces. Connect the EEPROM to the Arduino using the appropriate wires and pins. For I2C, you typically connect the SDA (data) and SCL (clock) pins, and for SPI, you'd connect MOSI (Master Out Slave In), MISO (Master In Slave Out), SCK (Serial Clock),

and possibly a chip-select (CS) pin. We have written the Arduino code using the Arduino Integrated Development Environment (IDE). We can read and write data to/from the EEPROM using the library functions provided.

8. CODE

/*External EEPROM Recording & Playback Demo*/

```
// Include the 12C Wire Library
#include<Wire.h>

// Include the Servo Library
#include<Servo.h>

// EEPROM 12C Address
#define EEPROM_I2C_ADDRESS 0x50

// Analog pin for potentiometer
int analogPin = 0;

// Integer to hold potentiometer value
int val = 0;

// Byte to hold data read from EEPROM
int readVal = 0;

// Integer to hold the number of addresses to fill
int maxaddress = 1500;
```

```
//Create a Servo object
Servo myservo;

// Function to write to EEPROM
void writeEEPROM(int address,byte val, int i2c_address)
{
// Begin transmission to 12C EEPROM
Wire.beginTransmission(i2c_address);

// Send memory address as two 8-bit bytes
Wire.write((int)(address >> 8)); //MSB
Wire.write((int)(address & 0xFF)); //LSB

// Send data to be stored
Wire.write(val);

// End the transmission
Wire.endTransmission();

// Add 5ms delay for EEPROM
delay (5);
}

//function to read from EEPROM
byte readEEPROM(int address,int i2c_address)

{
```

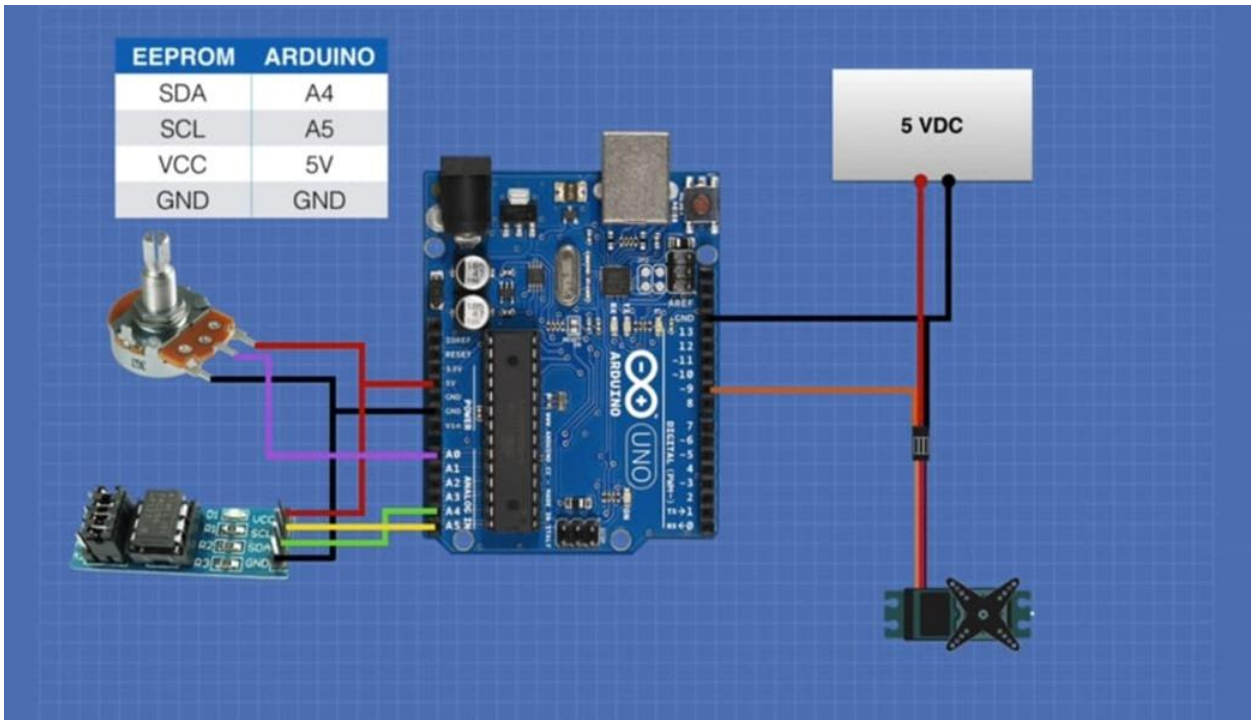
```
// Define byte for received data
byte rcvData = 0xFF;



// Begin transmission to 12C EEPROM
Wire.beginTransmission(i2c_address);

// Send memory address as two 8-bit bytes
Wire.write((int)(address>> 8)); // MSB
Wire.write((int) (address & 0xFF)); // LSB

// End the transmission
Wire.endTransmission();

// Request one byte of data at the current memory address
Wire.requestFrom(i2c_address, 1);

// Read the data and assign it to variable
rcvData = Wire.read();
// Return the data as function output
return rcvData;
}

void setup()
{
// Connect to 12C bus as master
Wire.begin();

// Setup Serial Monitor
```

```
Serial.begin(9600);

// Attach servo on pin 9 to the servo object
myservo.attach(9);
// Print to Serial Monitor
Serial.println("Start Recording...");

// Run until the maximum address is reached

for (int address = 0; address <= maxaddress; address++) {

// Read pot and map to range of 0.180 for servo
val = map(analogRead(analogPin), 0, 1023, 0, 180);

// Write to the servo
// Delay to allow servo to settle in position
myservo.write(val);
delay(15);

// Record the position in the external EEPROM
writeEEPROM(address, val, EEPROM_I2C_ADDRESS);

// Print to Serial Monitor
Serial.print("ADDR = ");
Serial.print(address);
Serial.print("\t");
Serial.println(val);
}
```

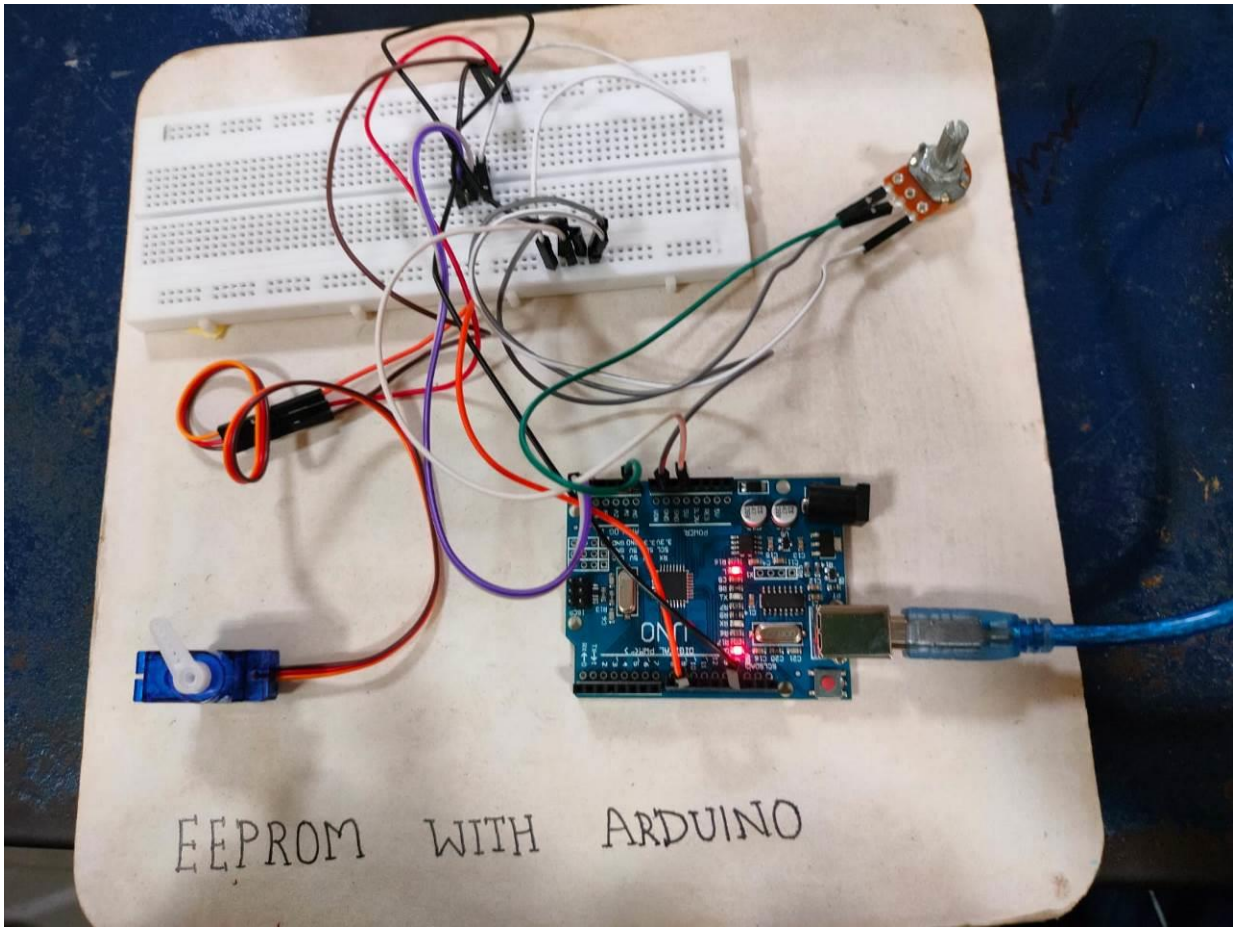
```
// Print to Serial Monitor
Serial.println("Recording Finished!");
// Delay 5 Seconds
delay(5000)
// Print to Serial Monitor
Serial.println("Begin Playback...")
// Run until maximum address is reached
for (int address = 0; address <= maxaddress; address++) {

// Read value from EEPROM
readVal = readEEPROM(address, EEPROM_I2C_ADDRESS);

// Write to the servo
// Delay to allow servo to settle in position
// Convert value to integer for servo

myservo.write(readVal);
delay(15);

// Print to Serial Monitor
Serial.print("ADDR = ");
Serial.print(address);
Serial.print("\t");
Serial.println(readVal);

}

// Print to Serial Monitor
Serial.println("Playback Finished!");
```

```
}

void loop() {

   // Nothing in loop
}
```

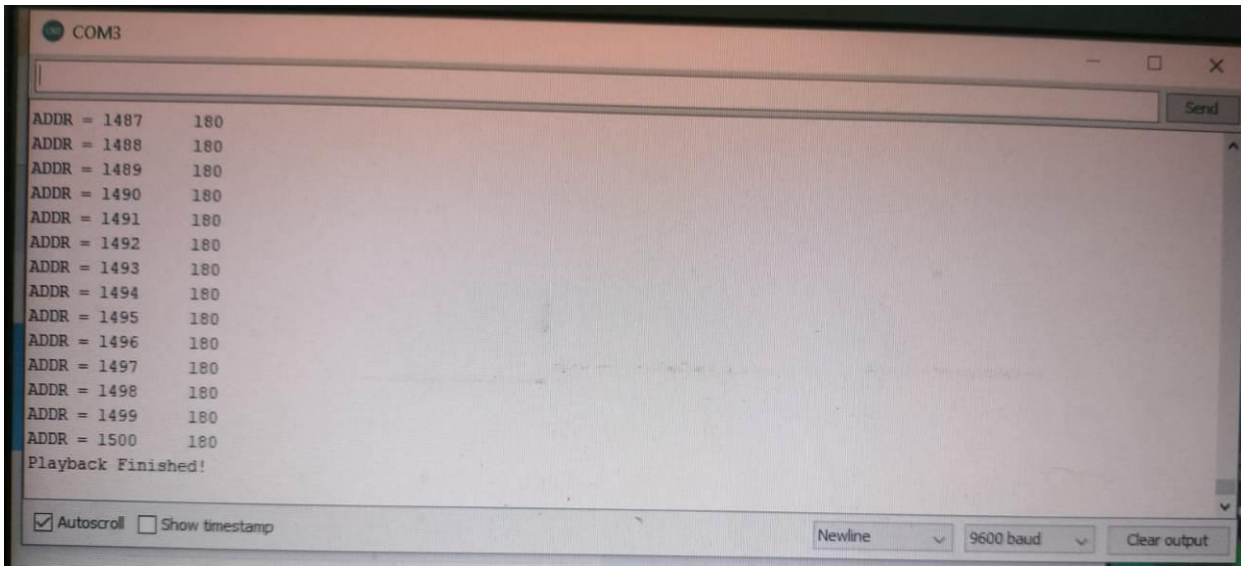
9. CIRCUIT DIAGRAM



10. PROJECT PHOTOS



11. STIMULATION RESULT



12. FUTURE ENHANCEMENT

1. Real-Time Clock (RTC) Integration: Incorporating an RTC module allows your project to timestamp data entries, enabling you to log data with time and date information. This is particularly useful for applications requiring precise time-based records.

2. Wireless Connectivity: Add wireless communication modules such as Wi-Fi, Bluetooth, or LoRa to enable remote data retrieval and control. This can transform your project into a powerful IoT device for data monitoring and remote access.

3. Data Encryption and Security: Implement data encryption and access control mechanisms to enhance the security of stored data. This is especially important for applications handling sensitive or confidential information

4. Data Visualization: Integrate a display module (e.g., OLED, TFT) to provide real-time data visualization on the device itself. Alternatively, you can design a web interface or mobile app to remotely monitor and visualize the data.

5. SD Card or External Storage: Expand data storage capacity by adding an SD card module or external storage options like USB flash drives. This can accommodate larger datasets and long-term data storage requirements.

6. Battery Management: Incorporate power-saving features and low-power modes to extend the project's battery life for applications where continuous power may not be available.

7. Data Backup and Recovery: Develop mechanisms for data backup and recovery in case of unexpected data loss or EEPROM corruption. This adds resilience to your data logging system.

8. User Interface and Configuration: Create a user-friendly interface to configure data logging settings, such as sampling frequency, thresholds, and alarm triggers. This simplifies customization for different applications

9. Cloud Integration: Connect your project to cloud platforms like AWS, Azure, or Google Cloud for seamless data storage and analysis. This opens up possibilities for big data analytics and remote access.

10. Machine Learning and Predictive Analysis: Implement machine learning algorithms for data prediction and anomaly detection. This can be useful in applications where early detection of anomalies is critical.

11. Integration with External Sensors: Allow your project to interface with a variety of sensors (e.g., environmental, biomedical, industrial) to make it adaptable for different use cases.

12. Multi-Platform Support: Make your project compatible with different Arduino boards and microcontrollers, expanding its reach and usability.

13. Documentation and Support: Provide comprehensive documentation and support resources to help users effectively implement and customize the project for their specific needs.

13. CONCLUSION

This project demonstrates the utility of external EEPROM modules in enhancing data storage and security for Arduino-based systems. By physically separating data storage from the main microcontroller, it opens up new possibilities for applications that demand robust data management. In summary, this abstract introduces an Arduino project that harnesses external EEPROM to address the limitations of onboard storage, expanding storage capacity and improving data security. This project offers a versatile solution with broad implications across various domains, from IoT devices to industrial automation and beyond.

14. REFERENCES

- https://youtu.be/ShqvATqXA7g?si=W-G3T_fGFt0jYxAN
- http://www.google.com/image
- http://www.wikipedia.com
- https://www.electronicsforu.com/
- https://www.electronicsandyou.com/