Smart Street Lights using LDR

Session: 2022 – 2026

Submitted by:

Hamid Ali 2022-CS-67

Tabish Akhtar 2022-CS-78

Saqlain Mansab 2022-CS-80

Bilal Asif 2022-CS-99

Supervised by:

Syed Tehseen Ul Hassan Shah

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

Table of Contents

Abstract

This report explores an automated streetlight system designed to enhance energy efficiency and safety. The system integrates light-detecting sensors with IoT technology to dynamically control streetlights. It details the flow of data from the sensors to decision-making modules, enabling the automatic adjustment of light intensity. The report highlights the system's ability to conserve energy by dimming or brightening lights based on environmental conditions. It also discusses the communication protocols used for remote monitoring and control through MQTT and Firebase Realtime Database. Overall, this system showcases an innovative approach to efficient and adaptive street lighting

Project Photo:

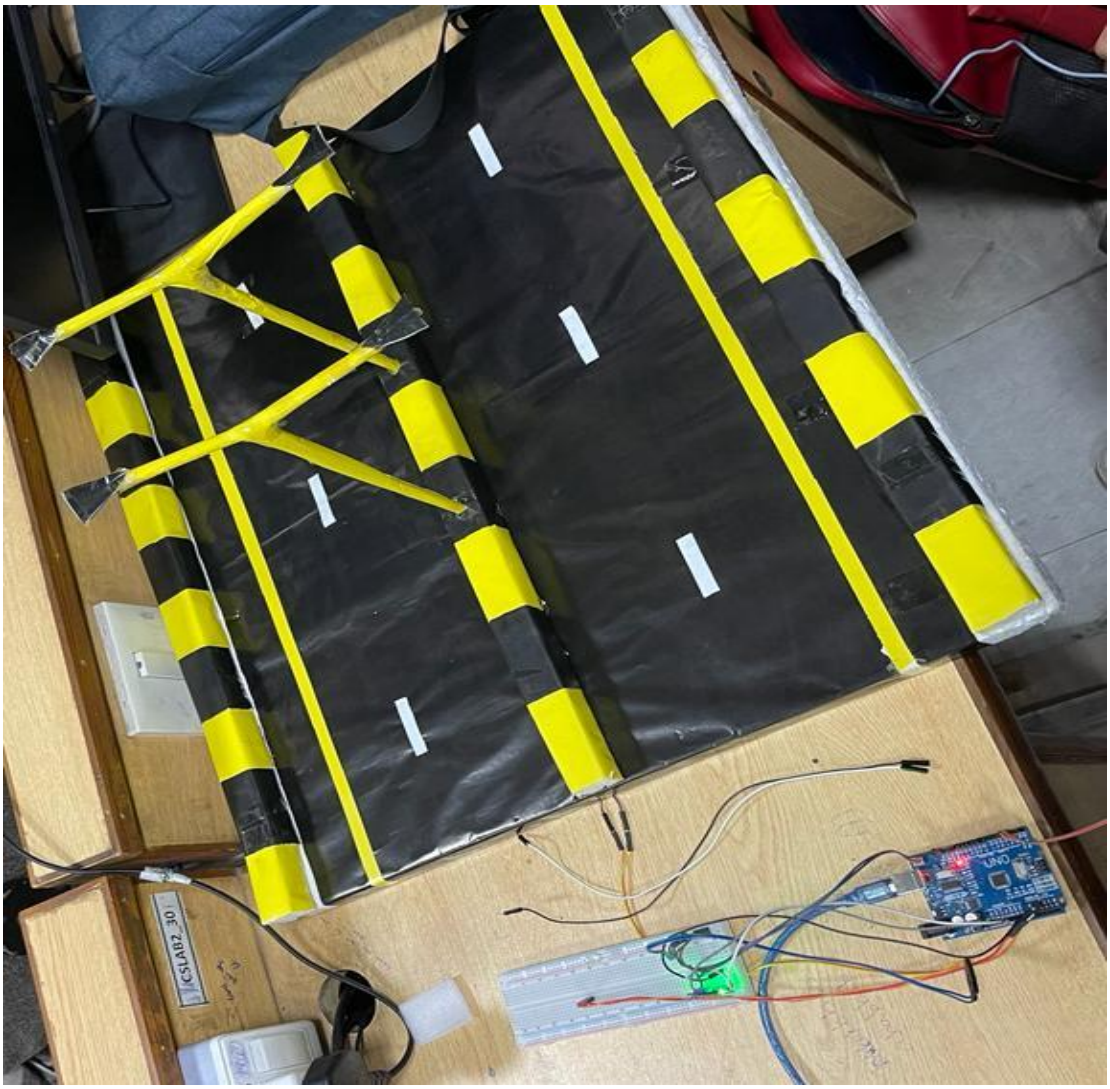


Figure 1: Project Model

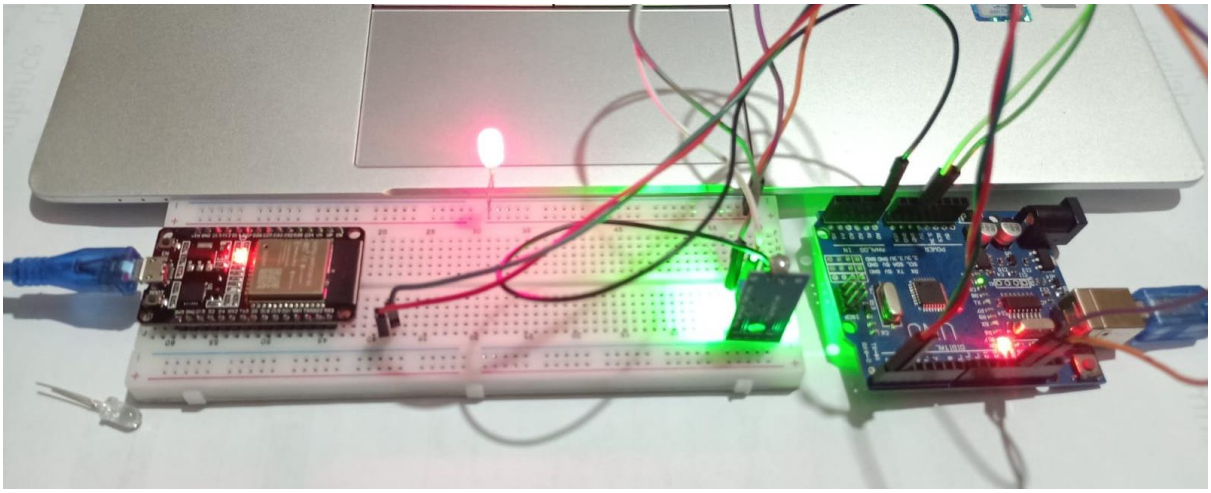


Figure 2: Components

Introduction:

Streetlights are essential for safety at night in cities. But many times, they use a lot of energy, and they don't adjust to the changing brightness outside. This report talks about a new kind of streetlight system. It's smart and can change how bright the lights are based on how much light is around.

This system uses special sensors and high-tech connections to make the lights work better. It's like giving the streetlights a brain to decide when to be bright and when to be dim. This report explains how all these parts fit together and make the streetlights work smarter.

We'll talk about how the sensors and computer parts talk to each other and how they make the lights change. This system is a big deal because it can save a lot of energy and make streets safer at the same time. The aim of this report is to show how this new streetlight system works, why it's important, and how it could make cities better places to live.

Objective:

- **Efficiency Enhancement:**

Implement smart controls to regulate street light intensity based on real-time environmental conditions, ensuring optimal energy usage.

- **Cost Reduction:**

Reduce electricity consumption and maintenance costs by implementing an automated system that only activates street lights when necessary.

- **Environmental Impact:**

Minimize the project's environmental footprint by integrating sensors that adjust lighting levels according to ambient light and traffic patterns.

- **Safety Improvement:**

Enhance public safety by ensuring well-lit streets during peak hours and dimming lights during off-peak times to conserve energy.

- **Data-driven Insights:**

Collect and analyze data from sensors to gain insights into usage patterns, aiding in future infrastructure planning and decision-making.

- **Remote Monitoring and Control:**

Enable remote monitoring and control of street lights via an IoT platform for efficient management and troubleshooting.

- **Integration and Scalability:**

Design a system that can be easily integrated with existing infrastructure and scaled up for future expansion or integration of additional smart city features.

Methodology:

Below is the methodology used in our project:

- **Requirement Analysis:**
  The project initiation involved a comprehensive analysis of requirements focused on creating smart, energy-efficient street lights. Identification of components such as the MQTT server, ESP32, Arduino Uno, Breadboard, White LEDs, Jumper Wires, and the LDR Module was based on the specific needs for connectivity, control, sensing, and illumination.
- **Component Selection and Integration:** Meticulous selection of components based on their functionalities, compatibility, and alignment with project objectives. Integration planning involved the MQTT server for centralizing control, ESP32 for connectivity, Arduino Uno as the processing unit, Breadboard for prototyping, White LEDs for illumination, Jumper Wires for connections, and the LDR Module as the light sensing element.
- **System Design and Prototyping:** The breadboard served as a platform to visualize and test connections between the selected components. This phase facilitated the creation of a prototype, allowing for rapid iteration, testing, and refinement of connections and functionalities among the Arduino Uno, LDR Module, and White LEDs.
- **Programming and Functional Testing:** Rigorous programming of the Arduino Uno to interact with the LDR Module was undertaken, ensuring proper functionality. Functional testing verified the system's ability to react to light intensity changes sensed by the LDR Module and appropriately control the White LEDs' illumination.
- **Integration with MQTT Dashboard:** Integration of the MQTT server and dashboard provided a user-friendly interface for monitoring, controlling, and setting threshold values remotely. This facilitated the visualization of real-time data from sensors and the ability to remotely adjust lighting levels or receive alerts based on anomalies detected by the system.

Components Overview:

1. **MQTT Server:**

The MQTT dashboard serves as our project's control center, providing a user-friendly interface to monitor and manage our smart street lights efficiently. MQTT (Message Queuing Telemetry Transport) is the technology that helps our devices, like the street lights, ESP32 and Arduino UNO, talk to each other and the dashboard. From this dashboard, Threshold value of light is set. This dashboard acts as a window into our street lights' behavior, allowing users to visualize real-time data collected from various sensors integrated into the lighting system. Through this dashboard, users can observe lighting patterns, environmental conditions, and energy usage metrics displayed in easy-to-understand graphs and charts. Additionally, it offers control functionalities, enabling users to remotely adjust lighting levels, set schedules, or receive alerts about any anomalies in the system. The MQTT dashboard's intuitive design and functionality empower users to make informed decisions, optimize energy usage, and ensure the optimal functioning of our smart street lights, contributing to a safer and more efficient urban environment.

2. ESP32:

Our smart street lights use a special technology called the ESP (Espressif Systems' Platform) to connect and work smartly. This ESP thing includes small but powerful computers called ESP32. They can easily connect the street lights to the internet, so they can talk to each other and share information in real-time. That IOT is programmed using Arduino IDE and its connected with Arduino UNO using transmit and receive pins. The ESP32 provides Wifi-connection

3. Arduino UNO:

The Arduino Uno is like the brain for our street lights. It's a small but clever computer that makes the lights work smartly. We use Microchip Studio IDE to program it. To connect the Arduino with the ESP, we use jumper wires that link their transmit and receive parts. When the Arduino gets light intensity info from the sensor, it checks this against a set level. If the light is brighter than this level, it turns on the LED. If the light is dimmer, the LED switches off. This helps the Arduino decide how bright the street lights should be based on what it sees, keeping things just right for us.

4. Breadboard:

The breadboard is a fundamental tool in our street lights project, acting as a platform to build and test electronic circuits without soldering. It's like a playground where we can connect different electronic components easily. Our project utilizes the breadboard to create and organize connections between various components, such as sensors, resistors, and wires, forming a prototype of our street lights system. Its grid-like layout with interconnected holes allows us to plug in components and create circuits swiftly.

5. White LED :

In our project, LEDs are strategically placed within the street light setup, providing bright and energy-efficient lighting. They're highly versatile and come in various colors, but for our purpose, we utilize white LEDs for optimal street illumination. LEDs are durable, have a longer lifespan, and produce minimal heat compared to traditional lighting sources, making them ideal for our smart street lights, where longevity and efficiency are crucial. Their ability to quickly turn on and off and adjust brightness levels allows our system to adapt to changing conditions, ensuring safety and energy conservation in our community.

6. Jumper Wires:

Jumper wires are essential connectors in our street lights project, acting as the 'cables' that link various components together on the breadboard or between devices like the Arduino Uno and ESP microcontroller. These wires, usually made of flexible, insulated material, allow for easy and temporary connections without the need for soldering. Their flexibility enables us to create connections between different points on the breadboard, establishing electrical pathways for data and power transmission between components. In our project, jumper wires facilitate the connection between the Arduino Uno and ESP, establishing communication between these key components. Their

convenience and versatility allow us to quickly prototype and adjust connections, enabling efficient testing and development of our street lights system. Jumper wires are instrumental in simplifying the circuit-building process, ensuring that our connections remain flexible and adaptable throughout the development stages.

7. **LDR Module:**

The LDR module, or Light Dependent Resistor module, is a vital sensor in our street lights system, designed to detect ambient light levels in the environment. This small electronic component changes its resistance based on the intensity of light falling on it. In our project, the LDR module serves as the 'eyes' of the street lights, providing information about the surrounding light conditions. As the amount of light changes, the resistance of the LDR also changes. This data is captured by the Arduino Uno, which then makes decisions about whether to turn the lights on or off based on the preset threshold level. When the light falls below this threshold, indicating darkness, the Arduino triggers the lights to turn on for safety and visibility. Conversely, when the light is brighter than the threshold, signifying daylight or sufficient illumination, the Arduino switches the lights off to conserve energy. The LDR module's sensitivity to light variations allows our street lights to adapt intelligently, ensuring optimal lighting conditions while conserving energy in our community.

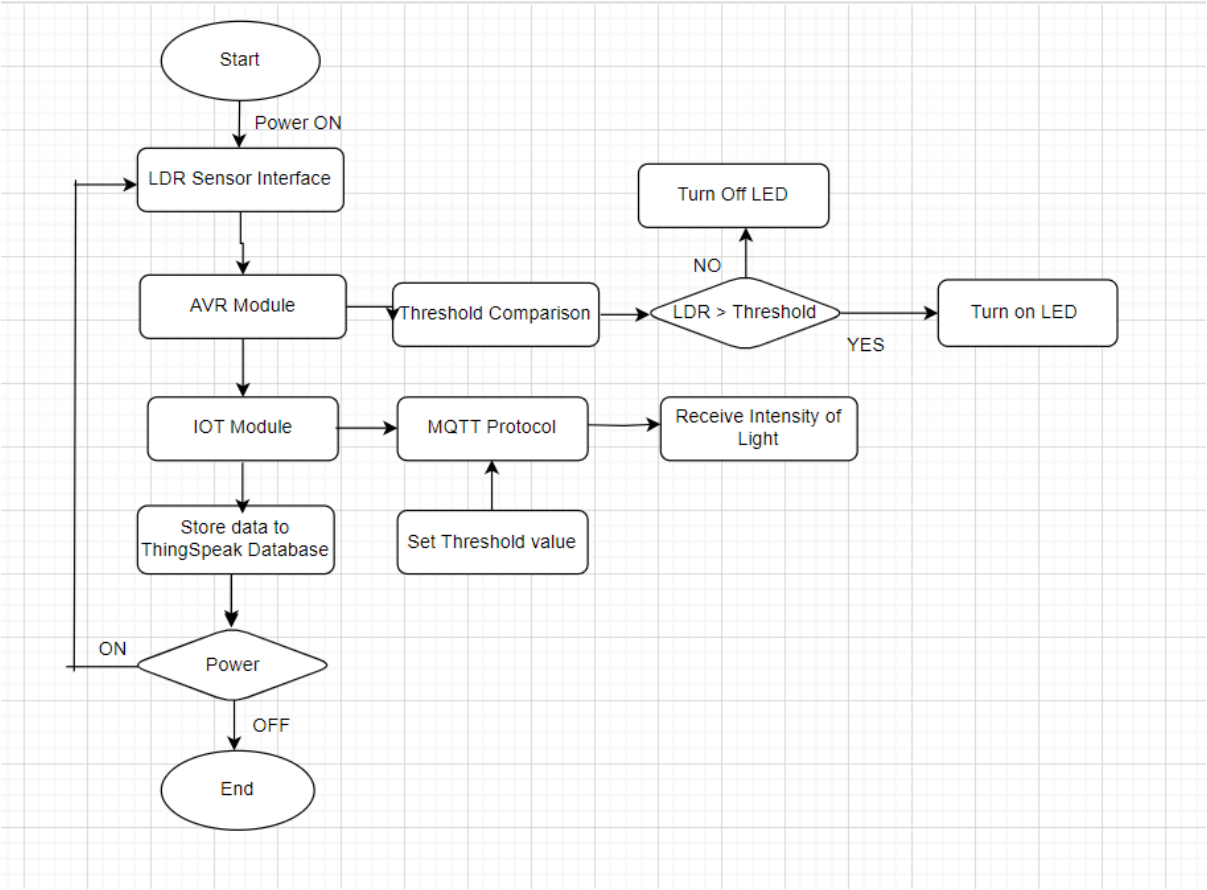
Functionalities:

Following are the some of functionalities of the project.

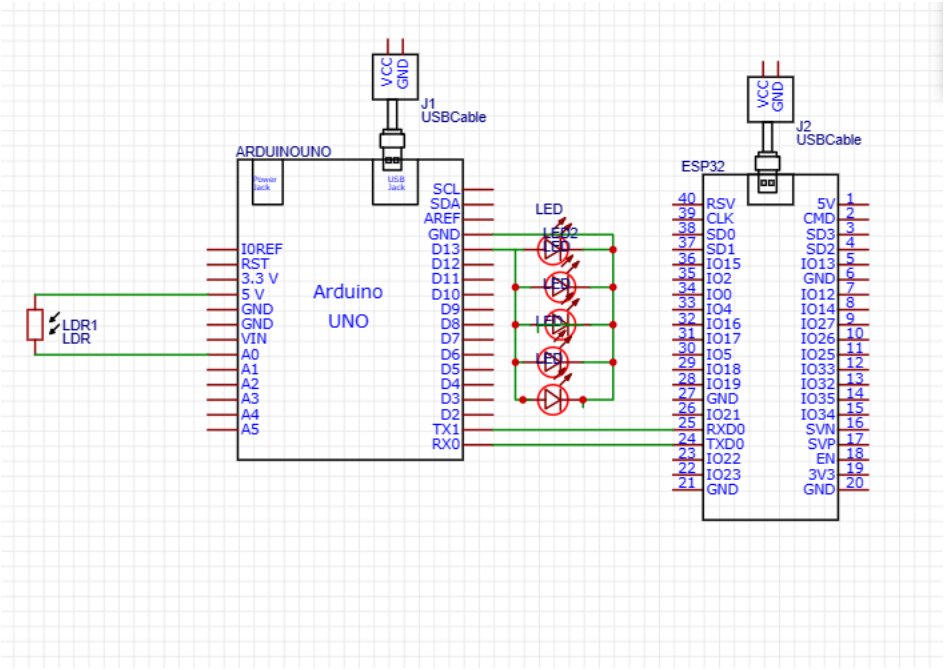
- **Automated Brightness Control:**
  The system utilizes sensors such as the LDR module to measure ambient light levels. Based on this data, the Arduino Uno adjusts the brightness of the LED lights automatically, ensuring optimal illumination for the prevailing environmental conditions.
- **Energy Conservation:**
  By dynamically adjusting light intensity, our street lights minimize unnecessary energy consumption. Lights dim during well-lit periods, such as daylight, and brighten up when ambient light decreases, contributing to significant energy savings.
- **Remote Monitoring and Control:**
  Through the MQTT dashboard, users can remotely monitor the status of the street lights, receiving real-time data on light intensity, power consumption, and system health. The dashboard also allows users to control lighting parameters, schedule on/off times, and receive alerts about any irregularities.
- **Adaptive Decision-Making:**
  The Arduino Uno, acting as the brain of the system, makes intelligent decisions based on data from sensors. It compares the received light intensity with predefined thresholds and dynamically adjusts the lighting, ensuring a responsive and adaptive lighting system.
- **Efficient Communication:**
  The MQTT server facilitates seamless communication between the various components of the system, including the Arduino Uno, ESP microcontroller, and the MQTT dashboard. This ensures reliable data exchange, enabling swift decision-making and control.

- **Rapid Prototyping with Breadboard:**
  During development, the breadboard serves as a versatile testing platform, allowing for quick and easy rearrangement of components. This facilitates rapid prototyping, troubleshooting, and experimentation with different configurations before final implementation.
- **User-Friendly Interface:**
  The MQTT dashboard provides a user-friendly interface for visualizing and controlling street light functionalities. Users can easily understand and interact with the system, making it accessible to a broad audience.

Data Flow Diagram:



Circuit diagram:



AVR Module Code:

```
.include "m328pdef.inc"

.include "macroDelay.inc"

.def A = r16

.def AH = r17

.def ThresholdAddr = r18

.org 0x00

    ldi r16, 0x44                ; Load lower 8 bits of baud rate
value   ---> 115200

     sts UBRR0L, r16

     ldi r16, 0x08                ; Load upper 8 bits of baud rate
value

     sts UBRR0H, r16

     ldi r16, (1 << TXEN0) | (1 << RXEN0)                ; Enable
transmitter and receiver
```

```
      sts UCSR0B, r16

      ldi r16, (1 << UCSZ01) | (1 << UCSZ00)              ; Set
frame format: 8 data bits, no parity, 1 stop bit

      sts UCSR0C, r16

      ; I/O Pins Configuration

      SBI DDRB,5          ; Set PB5 pin for Output to LED

      CBI PORTB,5         ; LED OFF

      ; ADC Configuration

      LDI A,0b11000111    ; [ADEN ADSC ADATE ADIF ADIE ADIE ADPS2
ADPS1 ADPS0]

      STS ADCSRA,A

      LDI A,0b01100000    ; [REFS1 REFS0 ADLAR – MUX3 MUX2 MUX1
MUX0]

      STS ADMUX,A         ; Select ADC0 (PC0) pin

      SBI PORTC,PC0       ; Enable Pull-up Resistor

      loop:

            LDS A,ADCSRA        ; Start Analog to Digital Conversion

            ORI A,(1<<ADSC)

            STS ADCSRA,A

      wait:

            LDS A,ADCSRA        ; wait for conversion to complete

            SBRC A,ADSC

            RJMP wait

            LDS A,ADCL          ; Must Read ADCL before ADCH

            LDS AH,ADCH

            delay 100           ; delay 100ms

            CP AH, r18          ; compare LDR reading with our
desired threshold
```

```
            BRSH LED_ON          ; jump if same or higher (AH >= 200)
            CBI PORTB,5          ; LED OFF
            ; Check if data is available in USART Data Register (UDR)
            lds r16, UCSR0A
            sbrs r16, RXC0
           rjmp loop ; Keep checking if no data
            ; Read received byte
            lds r18, UDR0
            ; Do something with the received byte (e.g., store in
memory)
            ;sts 0x100, r17 ; Store in memory address 0x100
            rjmp loop
      LED_ON:
            SBI PORTB,5          ; LED ON
      rjmp loop
```

IOT Module Code:

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ThingSpeak.h>
// Details of WIFI Connection
const char *ssid = "Saqlain";
const char *password = "11223344";
// Details of ThingSpeak Channel
long myChannelNumber = 2390848;
const char *myWriteAPIKey = "QYFPMA31BRT11Z26";
// long myChannelNumber = 2393185;
```

```
// const char* myWriteAPIKey = "RJSFHETTAVCOKY24";

// Details of MQTT Broker and topics

// const char *mqttServer = "test.mosquitto.org";

const char *mqttServer = "broker.hivemq.com";

const int mqttPort = 1883;

const char *mqttClientId = "2022CS80/device";

const char *inTopic = "2022-CS-80/value";      // threshold value
will be taken here

const char *outTopic = "2022-CS-80/intensity"; // intensity of
current light will be published here

// Random Variables

long currentTime, lastTime;

int count = 0;

char messages[50];

WiFiClient thingSpeakClient;

WiFiClient espClient;

PubSubClient client(espClient);

void callback(char *topic, byte *payload, unsigned int length){

    char buffer[length + 1]; // Add one for the null terminator

    memcpy(buffer, payload, length);

    buffer[length] = '\0'; // Null-terminate the string

    // Convert the payload to an integer

    int receivedValue = atoi(buffer);

    // Serial.print("Received Integer Value: ");

    Serial2.println(receivedValue);

}

void reconnect(){
```

```
    // Loop until we're reconnected
    while (!client.connected()){
        Serial.print("Attempting MQTT connection...");
        if (client.connect(mqttClientId)){
            Serial.println("connected");
            // Once connected, you can subscribe or publish to
topics
            client.subscribe(inTopic);
        }
        else{
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}
void setup(){
    Serial.begin(115200);
    Serial2.begin(115200);
    // Connect to Wi-Fi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED){
        delay(1000);
        Serial.println("Connecting to WiFi...");
```

```
    }
    Serial.println("Connected to WiFi");
    ThingSpeak.begin(thingSpeakClient);
    // Setup MQTT Server
    client.setServer(mqttServer, mqttPort);
    client.setCallback(callback);
    client.subscribe(inTopic);
    // Just connect it once
    while (!client.connected()){
        reconnect();
    }
}
void loop(){
    client.loop();
    // Check if data is available on Serial
    if (Serial.available() > 0){
        // Read the LDR value from Serial
        // After a value is read from Serial
        // 1. Display it on Serial Monitor
        // 2. Save it in ThingSpeak Database
        // 3. Publish it on MQTT broker

        int ldrValue = Serial.parseInt();
        while (Serial.available() > 0){
            Serial.read();
        }
```

```
        // Print the received value on Serial Monitor
        Serial.println("Received LDR Value: " + String(ldrValue));
        // Send data to ThingSpeak
        ThingSpeak.writeField(myChannelNumber, 1, ldrValue,
myWriteAPIKey);
        // delay(1000);      // Instead of delay we will use
millis() function to check time, we should not halt the program
        currentTime = millis();
        if (currentTime - lastTime > 500){
            snprintf(messages, 10, "%ld", ldrValue);
            // Serial.print("Loop Executed ");
            // Serial.println(messages);
            client.publish(outTopic, messages);
            lastTime = millis();
        }
    }
}
```

Code Documentation:

- # IoT Code Documentation

Libraries Used

WiFi.h: Library for connecting to WiFi networks.

PubSubClient.h: Library for MQTT communication.

Global Variables

ssid: Wi-Fi network's SSID (name).

password: Wi-Fi network password.

mqttServer: MQTT broker server address.

mqttPort: Port number for MQTT communication.

mqttClientId: Unique ID for the MQTT client.

mqttTopic: MQTT topic to subscribe to.

Objects Created

espClient: WiFiClient object for MQTT connection.

client: PubSubClient object for MQTT communication using espClient.

Functions

reconnect(): Reconnects to the MQTT broker if disconnected, continuously retries.

callback(): Callback triggered on receiving a message on the subscribed topic, displays the message payload.

setup_wifi(): Connects to Wi-Fi network using provided credentials.

setup_mqtt(): Sets up MQTT by specifying the server and callback function.

setup(): Arduino setup function initializing serial communication, connecting to Wi-Fi, setting up MQTT, and subscribing to the specified topic.

loop(): Arduino loop function ensuring MQTT connection is maintained and handling incoming messages.

Usage

Update ssid, password, mqttServer, mqttPort, mqttClientId, and mqttTopic with your network and MQTT broker details.

Ensure proper setup by calling setup_wifi() and setup_mqtt() functions in the setup() function.

Subscribe to the MQTT topic within setup() using client.subscribe(mqttTopic).

Customize callback() to process received messages.

Notes

Ensure the ESP32 board is correctly connected and powered for effective code execution.

- # AVR Code Documentation

Included Files

m328pdef.inc: Contains definitions specific to the ATmega328P microcontroller.

macroDelay.inc: Includes macros for implementing delays in the code.

UART Configuration

Configures UART (Universal Asynchronous Receiver-Transmitter) communication.

Sets baud rate to 115200, enables transmitter and receiver, and sets frame format (8 data bits, no parity, 1 stop bit).

I/O Pins Configuration

Sets PB5 as an output pin to control the LED.

Initially turns off the LED connected to PB5.

ADC Configuration

Sets up the ADC (Analog-to-Digital Converter) for LDR (Light Dependent Resistor) readings.

Enables ADC, selects ADC0 (PC0) pin, and sets a pull-up resistor on PC0.

Main Loop (loop:)

Initiates ADC conversion and waits for it to complete.

Reads the ADC values (ADCL and ADCH) representing LDR readings.

Delays 100ms before further processing.

LDR Reading Comparison and LED Control

Compares the LDR reading with a threshold value (stored in ThresholdAddr register) for LED control.

Turns the LED on if the LDR reading is equal to or higher than the threshold; otherwise, turns it off.

UART Data Reception

Checks for available data in the USART Data Register (UDR) and reads received byte if available.

Performs actions (e.g., stores the received byte in memory) with the received byte.

Usage

Modify the baud rate, UART settings, I/O configurations, and ADC setup as needed for your specific application.

Adjust the LED control logic based on your threshold requirements and LDR sensitivity.

Customize the actions for received UART data within the code section labeled Do something with the received byte.

Notes

This code assumes the setup of UART communication, ADC, LDR sensor, and LED connections on the ATmega328P microcontroller.

Project video links

LinkedIn:

Tabish Akhtar:

https://www.linkedin.com/feed/update/urn:li:activity:7148404651095232512/

Saqlain Mansab:

https://www.linkedin.com/feed/update/urn:li:activity:7148425529858142209/

Bilal Asif:

https://www.linkedin.com/posts/muhammad-bilal-asif-57b587269_syedtehseenulhasan-iot-smarttechnology-activity-7148564224556929024-l3NH?utm_source=share&utm_medium=member_desktop

Hamid Ali:

https://www.linkedin.com/posts/hamid-ali-76941a269_iotproject-smartcities-**innovationintech-activity-7148573608649428992-nmMC?utm_source=share&utm_medium=member_desktop**

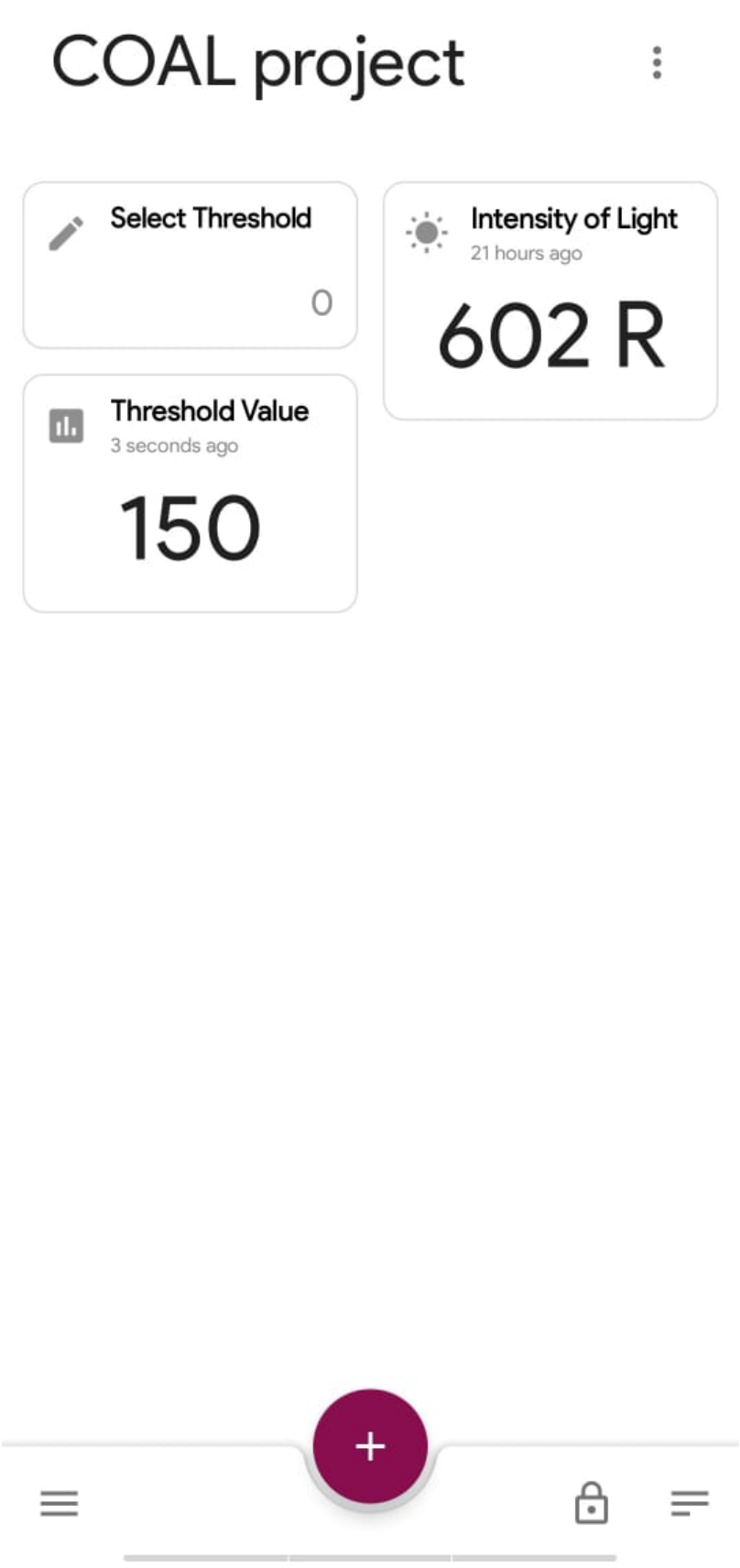
YouTube:

https://www.youtube.com/watch?v=OY8Fa3wQ0Xo

GitHub Link of project codes:

https://github.com/saqlainrai/COAL-Final-Project.git

MQTT Dashboard Screenshot:



Conclusion:

Our project, building smart street lights, combines smart technology and careful choices. With components like the MQTT server, ESP32, and Arduino Uno, we've made lights that can adapt to different light levels automatically. The dashboard we've created lets us see real-time data and control the lights from anywhere. It's not just about making streets brighter; it's about using less energy, making neighborhoods safer, and showing how technology can improve our cities. By carefully selecting and integrating these parts, we've built a system that responds to the environment, ensuring efficient and safe illumination while setting the stage for smarter, more sustainable urban living.

Future improvements:

Looking ahead, our smart street lights project holds potential for further enhancements and developments. Future iterations could explore the integration of advanced sensors, like motion or sound detectors, to enable more precise and responsive lighting adjustments based on specific environmental cues. Incorporating machine learning algorithms could empower the system to learn and adapt dynamically to evolving patterns, optimizing lighting strategies over time.

Additionally, exploring renewable energy sources, such as solar power, for our street lights could lead to greater sustainability and reduced dependency on traditional power grids. Enhancements in communication protocols could facilitate seamless integration with smart city networks, enabling data exchange between various urban systems for more comprehensive city planning.

Moreover, user-centric improvements in the dashboard interface could provide more granular control, analytics, and predictive maintenance insights, further empowering users to optimize lighting strategies and energy consumption.

Continued collaboration with urban planners, environmental experts, and community stakeholders will be essential to align future developments with the evolving needs of our neighborhoods, ensuring that our smart street lights not only illuminate our streets but also contribute to safer, greener, and more connected communities.

References

1. GeekforGeek,Intro to MQTT Protocol [online] https://www.geeksforgeeks.org/introduction-of-message-queue-telemetry-transport-protocol-mqtt/
2. GeekforGeek, Encapsulating Security Payload [online] https://www.geeksforgeeks.org/what-is-encapsulating-security-payload/
3. Connect ESP with MQTT, [online] https://www.makeuseof.com/using-esp01-with-mqtt-how-to-connect-and-control-iot-devices/
4. Serial connection B/W ESP and Arduino, [online] https://www.instructables.com/Serial-Communication-Between-Arduino-and-ESP-01/
5. Connect ESP to ThinkSpeak Cloud, https://www.electronicshub.org/connect-esp8266-to-thingspeak/