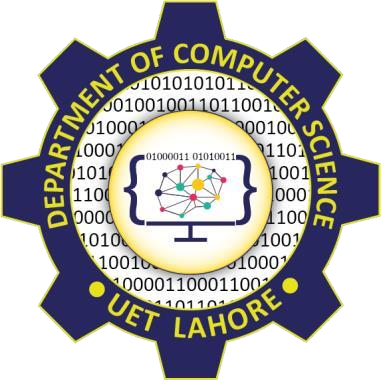
**Home Automation System**



Session: 2022 – 2026

**Submitted by:**

Rida Mushtaq 2022-CS-168

Asim Ali Murtaza 2022-CS-175

Rafiya Rehan 2022-CS-182

Ayesha 2022-CS-213

**Supervised by:**

Sir Tehseen Ul Hassan Shah

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**Contents**

[**Project Picture** 5](#_Toc155133232)

[**Project Description** 6](#_Toc155133233)

[**AVR Module** 6](#_Toc155133234)

[**Fan Speed Control** 6](#_Toc155133235)

[**Light Control** 6](#_Toc155133236)

[**Smartphone Integration** 6](#_Toc155133237)

[**Light Intensity Detection** 6](#_Toc155133238)

[**IoT Module** 6](#_Toc155133239)

[**MQTT Dash App Integration** 6](#_Toc155133240)

[**Fan Speed Adjustments** 6](#_Toc155133241)

[**Cloud Integration** 6](#_Toc155133242)

[**Real-time Updates** 6](#_Toc155133243)

[**Methodology Used** 7](#_Toc155133244)

[**Requirements Analysis** 7](#_Toc155133245)

[**Hardware Selection** 7](#_Toc155133246)

[**Programming Languages** 7](#_Toc155133247)

[**Fan Speed Control Implementation** 7](#_Toc155133248)

[**Light Control Implementation** 7](#_Toc155133249)

[**Smartphone Integration** 7](#_Toc155133250)

[**Light Intensity Resistor (LDR)** 7](#_Toc155133251)

[**MQTT Dash App Integration for IoT Module** 7](#_Toc155133252)

[**Cloud Integration** 7](#_Toc155133253)

[**Real-time Updates** 7](#_Toc155133254)

[**System Integration and Testing** 8](#_Toc155133255)

[**Data Flow Diagram** 8](#_Toc155133256)

[**Flow-Chart** 9](#_Toc155133257)

[**Circuit Diagram** 10](#_Toc155133258)

[**Working of Components** 11](#_Toc155133259)

[**ATmega328P Microcontroller (Programmed in AVR Assembly)** 11](#_Toc155133260)

[**Light and Fan Control** 11](#_Toc155133261)

[**LDR Module Integration** 11](#_Toc155133262)

[**UART Communication** 11](#_Toc155133263)

[**Assembly Language Programming** 11](#_Toc155133264)

[**ESP32 WROOM Dev Board (Programmed in C++ using Arduino IDE)** 11](#_Toc155133265)

[**Wi-Fi Communication** 11](#_Toc155133266)

[**MQTT Protocol Implementation** 11](#_Toc155133267)

[**UART Communication with ATmega328P** 12](#_Toc155133268)

[**ThingSpeak Integration** 12](#_Toc155133269)

[**C++ Programming using Arduino IDE** 12](#_Toc155133270)

[**LDR Module (Light Dependent Resistor)** 12](#_Toc155133271)

[**Light Sensing** 12](#_Toc155133272)

[**Analog Signal Output** 12](#_Toc155133273)

[**Mobile Phone (User Interface)** 12](#_Toc155133274)

[**User Commands** 12](#_Toc155133275)

[**Wi-Fi Communication** 12](#_Toc155133276)

[**Real-time Feedback** 12](#_Toc155133277)

[**ThingSpeak Server** 13](#_Toc155133278)

[**Data Upload** 13](#_Toc155133279)

[**Visualization** 13](#_Toc155133280)

[**Analysis and Insights** 13](#_Toc155133281)

[**AVR Module Code** 13](#_Toc155133282)

[**IOT Module Code** 17](#_Toc155133283)

[**Code Documentation** 22](#_Toc155133284)

[**AVR Include Files** 22](#_Toc155133285)

[**Macros Used** 22](#_Toc155133286)

[**Labels** 22](#_Toc155133287)

[**ESP32 Libraries Used** 23](#_Toc155133288)

[**MQTT Dash app’s dashboard screenshots** 24](#_Toc155133289)

[**Video Links** 25](#_Toc155133290)

[**GitHub Link** 25](#_Toc155133291)

[**References** 25](#_Toc155133292)

Table of figures:

[Figure 1:Project Picture 5](file:///D:\UET%20BSCS-22\3rd%20Semester\Computer%20Organization%20and%20Assembly%20Language\Lab\CoalReport.docx#_Toc155283951)

[Figure 2: Data Flow Diagram 8](#_Toc155283952)

[Figure 3: Flow Chart Diagram 9](file:///D:\UET%20BSCS-22\3rd%20Semester\Computer%20Organization%20and%20Assembly%20Language\Lab\CoalReport.docx#_Toc155283953)

[Figure 4: Circuit Diagram 10](file:///D:\UET%20BSCS-22\3rd%20Semester\Computer%20Organization%20and%20Assembly%20Language\Lab\CoalReport.docx#_Toc155283954)

[Figure 5:Lights and Fans Control 26](file:///D:\UET%20BSCS-22\3rd%20Semester\Computer%20Organization%20and%20Assembly%20Language\Lab\CoalReport.docx#_Toc155283955)

[Figure 6: LDR Value 26](file:///D:\UET%20BSCS-22\3rd%20Semester\Computer%20Organization%20and%20Assembly%20Language\Lab\CoalReport.docx#_Toc155283956)

# **Project Picture**

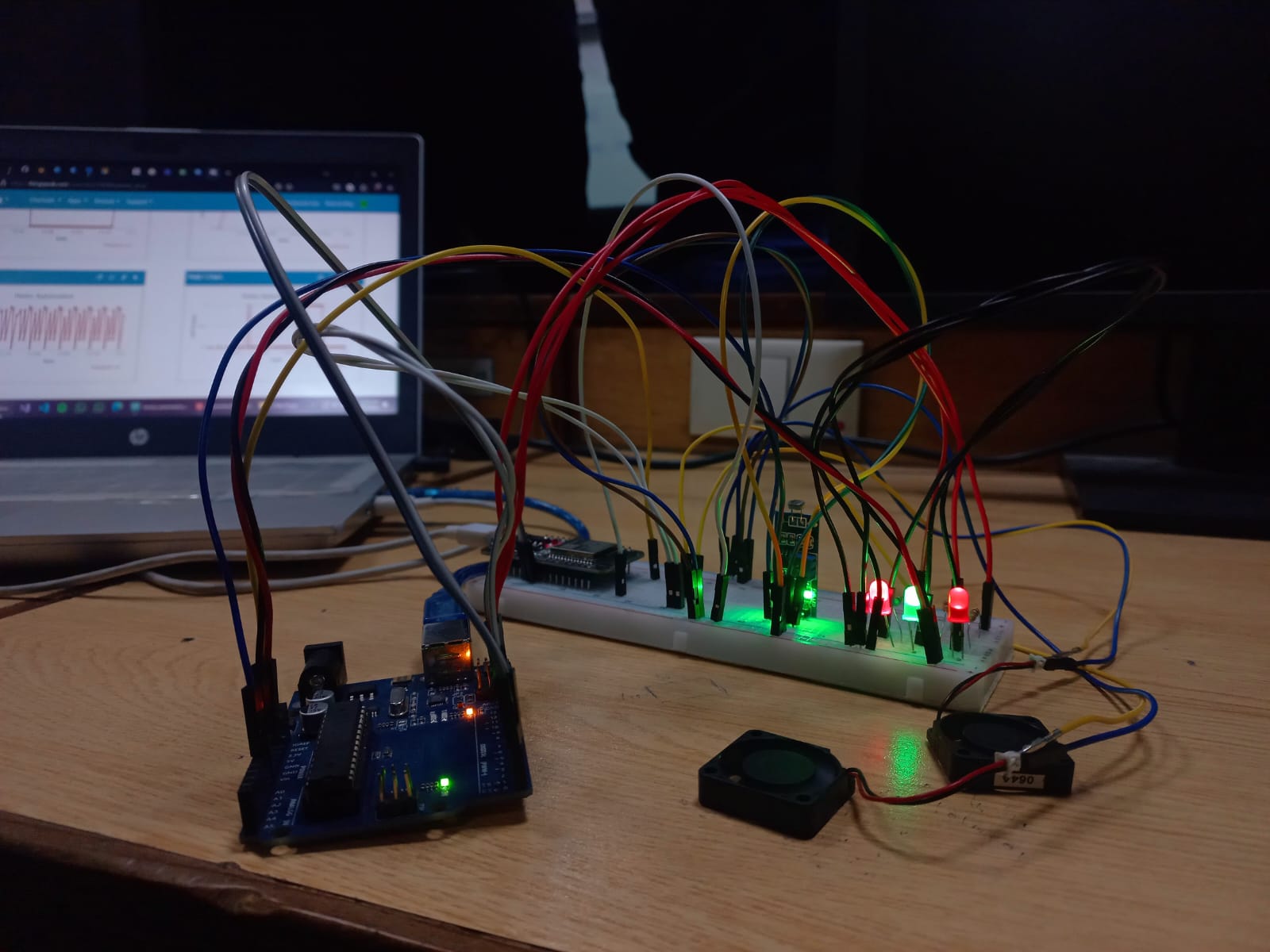


Figure 1:Project Picture

# **Project Description**

The Smart Home Automation System empowers users to remotely manage lighting and fan systems via a mobile phone. The system features an ATmega328P microcontroller programmed in AVR Assembly language for optimal control and an ESP32 WROOM development board programmed in C++ using the Arduino IDE for seamless Wi-Fi communication. Additionally, it offers the capability to vary the speed of one fan for enhanced customization.

## **AVR Module**

Brief explanation for AVR Module is given.

**Fan Speed Control**

Utilizes ATmega328P microcontroller with PWM pin for precise fan speed adjustments.

**Light Control**

Manages on/off status of three lights based on MQTT Dash app commands.

**Smartphone Integration**

Enables remote control through MQTT Dash app on user's smartphone.

**Light Intensity Detection**

Incorporates LDR sensor for ambient light intensity monitoring.

## **IoT Module**

Brief explanation for IOT Module is given.

**MQTT Dash App Integration**

ESP32 communicates with the MQTT Dash app for a user-friendly interface.

**Fan Speed Adjustments**

Range button for second fan on/off and speed control.

### **Cloud Integration**

Sends data (LDR values, device statuses) to ThingSpeak cloud for logging.

### **Real-time Updates**

Transmits status updates to the smartphone through MQTT protocol, ensuring real-time control and monitoring.

# **Methodology Used**

The methodology involves using Assembly language for the ATmega328P and C++ for the ESP32 within the Arduino IDE. The ATmega328P is responsible for controlling lights, fans, and processing data from the LDR module. The ESP32 handles Wi-Fi communication, MQTT protocol implementation, and data transfer to the ThingSpeak server.

## **Requirements Analysis**

User requirements for the Home Automation System, considering features such fan speed customization, light management, and real-time updates are analyzed.

## **Hardware Selection**

Appropriate hardware components, including the ATmega328P microcontroller for the AVR module and the ESP32 WROOM development board for the IoT module, based on their compatibility with the project requirements are used. LEDs operate on 3Volts and fans work on 5Volts.

## **Programming Languages**

AVR Assembly language is used for programming the ATmega328P microcontroller. For the ESP32 WROOM development board, C++ in the Arduino IDE for seamless Wi-Fi communication is employed.

## **Fan Speed Control Implementation**

Fan speed is controlled using the PWM pin of the ATmega328P microcontroller, enabling precise adjustments.

## **Light Control Implementation**

The ATmega328P microcontroller is programmed to manage the on/off status of the three lights based on commands received from the MQTT Dash app.

## **Smartphone Integration**

Communication is established between the Smart Home Automation System and the MQTT Dash app on the user's smartphone for seamless remote control.

## **Light Intensity Resistor (LDR)**

The LDR sensor is incorporated into the system.

## **MQTT Dash App Integration for IoT Module**

Communication protocols are developed between the ESP32 WROOM development board and the MQTT Dash app. User-friendly interface with buttons for light and fan control, as well as a range button for fan speed adjustments is employed.

## **Cloud Integration**

Functionalities to send data, including LDR values and device statuses, to the ThingSpeak cloud platform for logging and monitoring purposes are implemented.

## **Real-time Updates**

The system is configured to transmit real-time updates on device statuses to the user's smartphone through the MQTT protocol.

## **System Integration and Testing**

AVR and IoT modules are integrated into Home Automation System.

# **Data Flow Diagram**

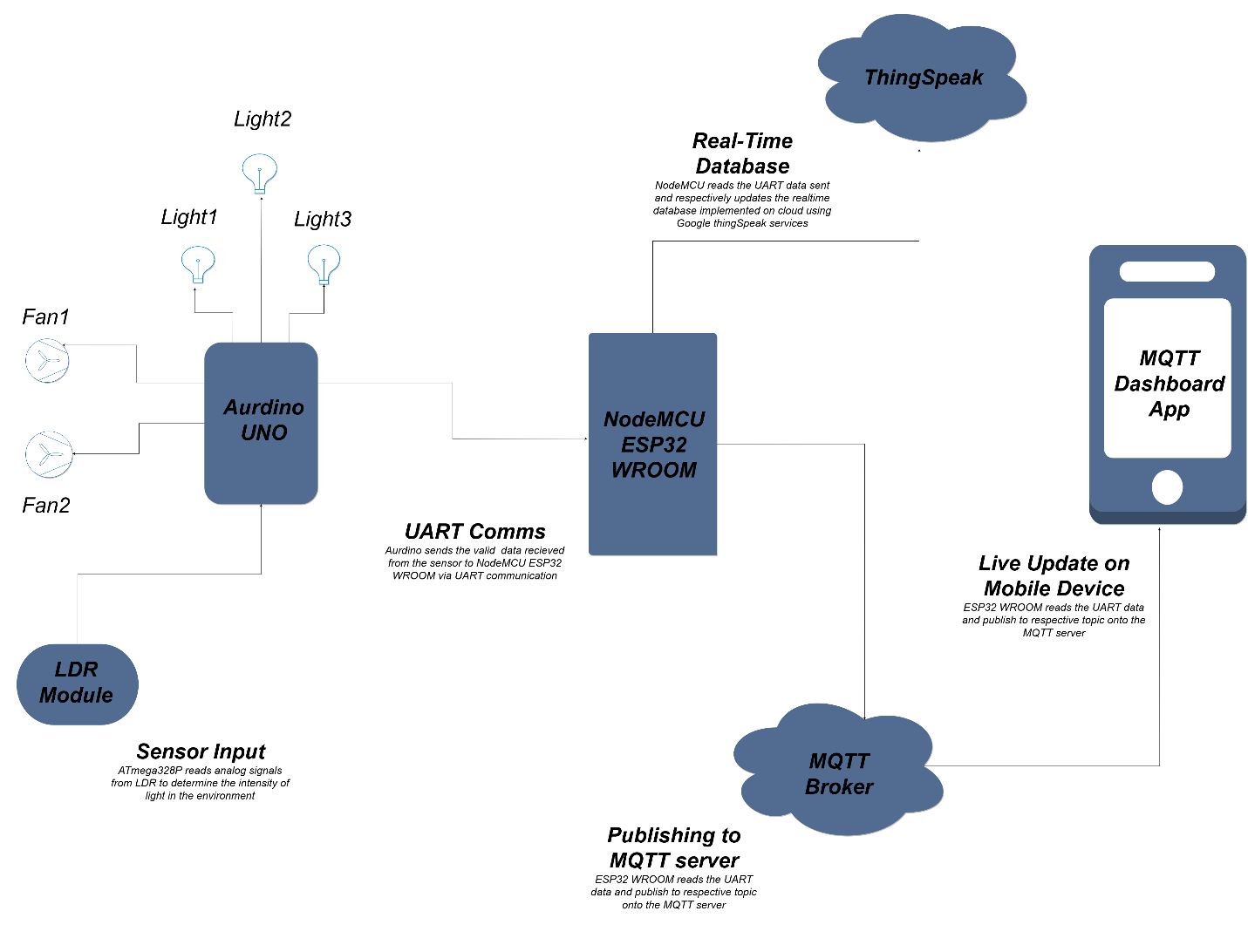


Figure 2: Data Flow Diagram

# **Flow-Chart**

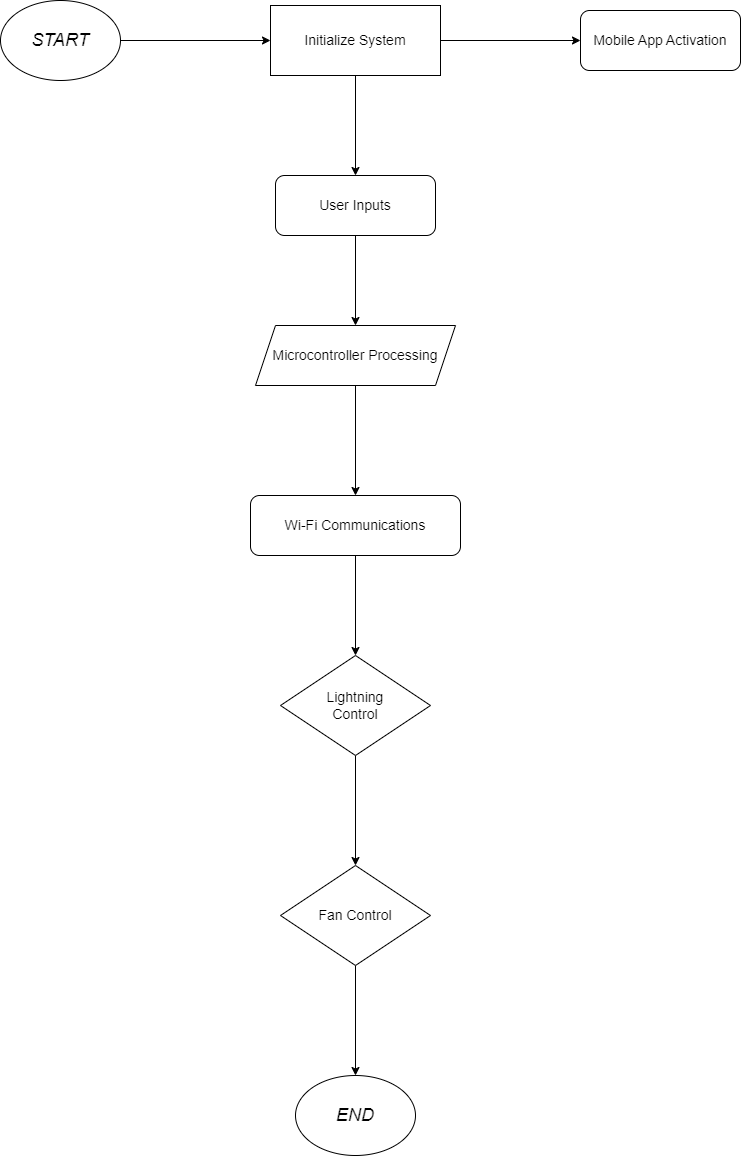


Figure 3: Flow Chart Diagram

# **Circuit Diagram**

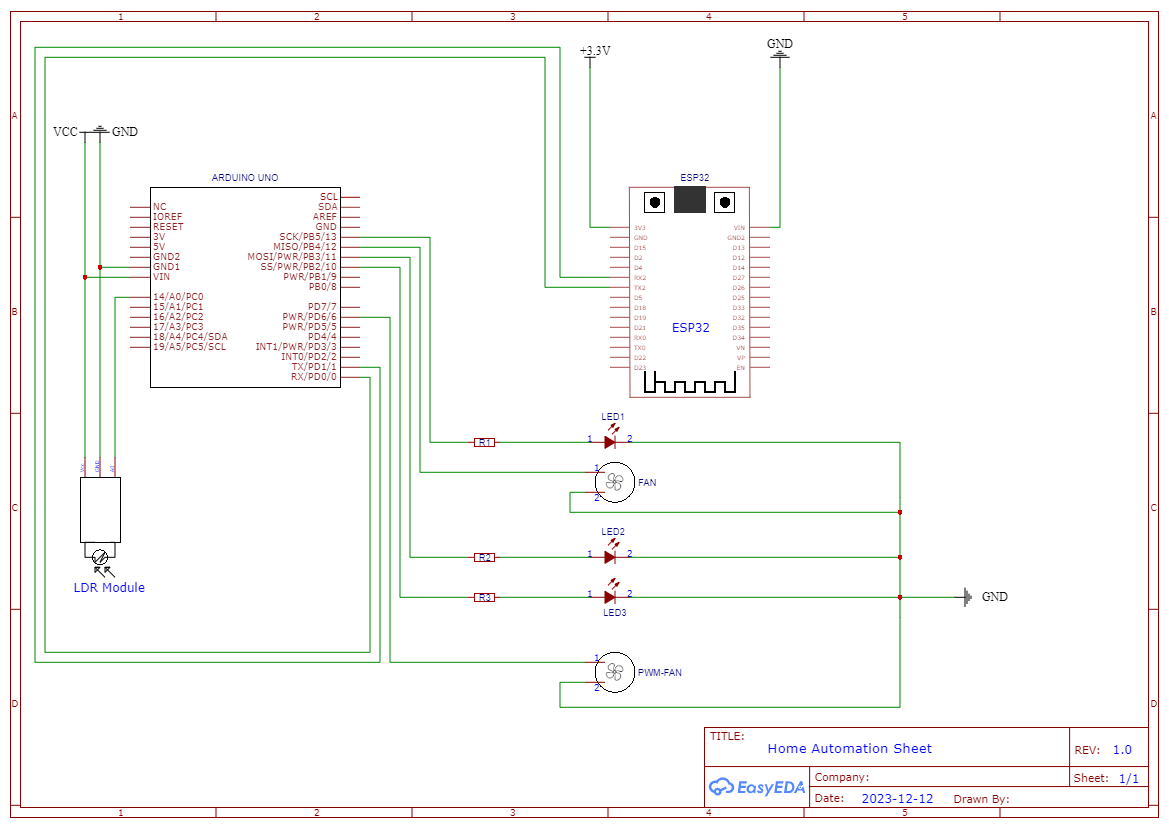


Figure 4: Circuit Diagram

# **Working of Components**

## **ATmega328P Microcontroller (Programmed in AVR Assembly)**

The ATmega328P microcontroller serves as the brain of the system, responsible for controlling lights, fans, adjusting fan speed and processing data from the LDR module. Further details are listed below.

### **Light and Fan Control**

* The microcontroller monitors input signals from the mobile application or external switches to determine user commands.
* Depending on the received commands, the ATmega328P toggles the state of the lights and fans by controlling corresponding output pins.
* One of the fans is connected to PWM (AVR port) and speed of this fan can vary by simply using a slider in MQTT. A range is provided to slider to adjust the speed.

### **LDR Module Integration**

* An LDR (Light Dependent Resistor) module is used to detect ambient light levels.
* The ATmega328P reads analog signals from the LDR to determine the intensity of light in the environment.

### **UART Communication**

* The ATmega328P communicates with the ESP32 via UART (Universal Asynchronous Receiver-Transmitter) for exchanging data.
* UART allows serial communication between the microcontroller and ESP32, enabling them to share control commands and sensor data.

### **Assembly Language Programming**

* The microcontroller is programmed in AVR Assembly language for efficient and low-level control.
* Assembly language allows for precise manipulation of hardware registers, making it suitable for resource-constrained microcontrollers like the ATmega328P.

## **ESP32 WROOM Dev Board (Programmed in C++ using Arduino IDE)**

The ESP32 WROOM development board handles Wi-Fi communication, MQTT protocol implementation, and data transfer to the ThingSpeak server.

## **Wi-Fi Communication**

* The ESP32 connects to a Wi-Fi network, allowing it to establish a communication link with the user's mobile phone and other networked devices.
* This enables remote control and monitoring of the Smart Home Automation system.

### **MQTT Protocol Implementation**

* MQTT (Message Queuing Telemetry Transport) is employed for communication between the ESP32 and ATmega328P.
* The ESP32 subscribes to specific topics for receiving control commands from the user's phone and publishes sensor data to the ATmega328P.

### **UART Communication with ATmega328P**

* The ESP32 communicates with the ATmega328P via UART to exchange control commands and sensor data.
* It sends user commands received over Wi-Fi to the ATmega328P and receives sensor data for further processing or transmission to the ThingSpeak server.

### **ThingSpeak Integration**

* The ESP32 uploads sensor data to the ThingSpeak server for analysis and visualization.
* This data can include information about light intensity from the LDR module, user commands, and other relevant information.

### **C++ Programming using Arduino IDE**

* The ESP32 is programmed in C++ using the Arduino IDE, which simplifies the development process and allows for easy integration of libraries for Wi-Fi and MQTT communication.

## **LDR Module (Light Dependent Resistor)**

The LDR module is used for detecting ambient light levels, enabling automation based on the intensity of light.

### **Light Sensing**

* The LDR (Light Dependent Resistor) changes its resistance based on the intensity of light falling on it.
* As ambient light levels change, the resistance of the LDR varies proportionally.

### **Analog Signal Output**

* The LDR module provides an analog signal that corresponds to the light intensity.
* This analog signal is read by the ATmega328P, allowing it to determine the current ambient light conditions.

## **Mobile Phone (User Interface)**

The mobile phone serves as the user interface, allowing users to remotely control lights and fans.

### **User Commands**

* Users interact with a mobile application to send commands to ESP32P.
* The application has buttons and sliders for controlling lights and fans.

### **Wi-Fi Communication**

* The mobile phone communicates with the ESP32 over a Wi-Fi network.
* Users can issue commands, and the ESP32 processes these commands to control the lights and fans through the ATmega328P.

### **Real-time Feedback**

* The mobile application may provide real-time feedback on the status of lights and fans.
* Users can receive information about the current state of the system and any changes made remotely.

## **ThingSpeak Server**

The ThingSpeak server is used for data analysis and visualization.

### **Data Upload**

* The ESP32 uploads sensor data, including light intensity information, to the ThingSpeak server.
* This data is sent periodically, allowing for continuous monitoring and analysis.

### **Visualization**

* ThingSpeak provides tools for visualizing data in the form of charts and graphs.
* Users can access the ThingSpeak platform to view historical trends and patterns related to light intensity and system activity.

### **Analysis and Insights**

* The uploaded data can be analyzed to derive insights into the performance of the Home Automation system.
* Patterns in light intensity, usage trends, and other metrics can be identified.

# **AVR Module Code**

.include "m328pdef.inc"

.include "delay.inc"

.include "UART\_Macros.inc"

.def A = r17

.def AH = r18

.def TEMP = r22

.def UART\_CHAR = r21

.org 0x0000 ; Reset vector

RJMP RESET\_handler ; Relative jump to the reset handler

.org 0x0024 ; USART RX interrupt vector

RJMP UART\_RX\_ISR ; Jump to USART ISR

RESET\_handler:

LDI r16, high(RAMEND) ;stack pointer initialization

OUT SPH, r16

LDI r16, low(RAMEND)

OUT SPL, r16

SEI

; Macro to read a single byte from the UART

; Inputs: register to hold the received byte

; Outputs: r16

; Working: receives byte via UART and stores in a r16 register

.macro Serial\_read

;Serial\_read\_WAIT:

; wait till a byte is in the buffer

LDS R25, UCSR0A

SBRS R25, RXC0

RJMP Serial\_read\_Skip

;RJMP Serial\_read\_WAIT

Serial\_read\_Start:

; read the byte

LDS r25, UDR0

rjmp Serial\_read\_END

Serial\_read\_Skip:

LDI r25,0

Serial\_read\_END:

.endmacro

main:

;-------------------------------- Configurations ---------------------------------

; I/O Pins Configuration

SBI DDRB,5 ; Set PB5 pin for Output to LED1

CBI PORTB,5 ; LED1 OFF

SBI DDRB,3 ; Set PB5 pin for Output to LED2

CBI PORTB,3 ; LED2 OFF

SBI DDRB,2 ; Set PB5 pin for Output to LED3

CBI PORTB,2 ; LED3 OFF

SBI DDRB,4 ; Set PB4 pin for Output to Fan1

CBI PORTB,4 ; Fan1 OFF

SBI DDRD,PD6 ; Set PD6 pin for Output to Fan2 PWM

SBI PORTD,PD6

;ADC Cofiguration

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

; PWM Configuration for Fan2 (PD6)

ldi TEMP,0xFF

out OCR0A,TEMP ; to compare match port A

ldi TEMP, (1 << COM0A1) | (0 << COM0A0) | (1 << WGM01) | (1 << WGM00)

out TCCR0A,TEMP

ldi TEMP,1<<CS00 ; Prescaler = 1

out TCCR0B,TEMP

; UART Configuration

SBI DDRD,1 ; Set PD1 (TX) as Output

CBI PORTD,1 ; TX Low (initial state)

CBI DDRD,0 ; Set PD0 (RX) as Input

SBI PORTD,0 ; Enable Pull-up Resistor on RX

Serial\_begin ; Initialize UART Protocol

;-------------------------LED Logic----------------------

loop:

LDS A, ADCSRA ; Start Analog to Digital Conversion

ORI A, (1 << ADSC)

STS ADCSRA, A

wait:

LDS A, ADCSRA

SBRC A, ADSC

RJMP wait

LDS A, ADCL ; Must Read ADCL before ADCH

LDS AH, ADCH

delay 100 ; delay 100ms

Serial\_writeReg AH ; sending the received value to UART

Serial\_writeChar ':' ; just for formating

Serial\_writeChar ' '

RJMP loop

;--------------------FANS Interrupt Logic------------------------

UART\_RX\_ISR:

LDI r25, 0

Serial\_read

CPI r25, 1 ;Fan 1

BREQ turn\_fan1\_on

CPI r25, 2

BREQ turn\_fan1\_off

CPI r25, 3 ;LED 1

BREQ turn\_light1\_on

CPI r25, 4

BREQ turn\_light1\_off

CPI r25, 5 ;LED 2

BREQ turn\_light2\_on

CPI r25, 6

BREQ turn\_light2\_off

CPI r25, 7 ;LED 3

BREQ turn\_light3\_on

CPI r25, 8

BREQ turn\_light3\_off

RJMP fan2\_speed

RJMP UART\_RX\_ISR\_End

turn\_fan1\_on:

SBI PORTB,4 ; Fan1 ON

RJMP UART\_RX\_ISR\_End

turn\_fan1\_off:

CBI PORTB,4 ; Fan1 OFF

RJMP UART\_RX\_ISR\_End

turn\_light2\_on:

SBI PORTB,3

RJMP UART\_RX\_ISR\_End

turn\_light1\_on:

SBI PORTB,5

RJMP UART\_RX\_ISR\_End

turn\_light3\_on:

SBI PORTB,2

RJMP UART\_RX\_ISR\_End

turn\_light1\_off:

CBI PORTB,5

RJMP UART\_RX\_ISR\_End

turn\_light2\_off:

CBI PORTB,3

RJMP UART\_RX\_ISR\_End

turn\_light3\_off:

CBI PORTB,2

RJMP UART\_RX\_ISR\_End

fan2\_speed:

OUT OCR0A,r25

RJMP UART\_RX\_ISR\_End

UART\_RX\_ISR\_End:

RETI ; Return from interrupt

# **IOT Module Code**

#include <WiFi.h>

#include <PubSubClient.h>

#include <HTTPClient.h>

// WiFi

const char \*ssid = "asimali"; // Enter your Wi-Fi name

const char \*password = "33103310"; // Enter Wi-Fi password

// MQTT Broker

const char \*mqtt\_broker = "test.mosquitto.org";

const char \*topic1 = "fan1/control";

const char \*topic2 = "fan2/speed";

const char \*topic3 = "light1/control";

const char \*topic4 = "light2/control";

const char \*topic5 = "light3/control";

const char \*topic6 = "LDRValue";

int LED1Status;

int LED2Status;

int LED3Status;

int fanStatus;

int fanSpeed;

int ldrValue;

int receivedValue = 0;

int count = 0;

const char \*mqtt\_username = "";

const char \*mqtt\_password = "";

const int mqtt\_port = 1883;

const long channelId = 2390484;

const char\* apiKey = "II2KTL4CZTQK4MH0";

WiFiClient espClient;

PubSubClient client(espClient);

void setup() {

Serial.begin(115200);

Serial2.begin(9600, SERIAL\_8N1, 16, 17); // RX2=16, TX2=17 on ESP32

// Connecting to a WiFi network

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.println("Connecting to WiFi..");

}

Serial.println("Connected to the Wi-Fi network");

//connecting to a mqtt broker

client.setServer(mqtt\_broker, mqtt\_port);

client.setCallback(callback);

while (!client.connected()) {

String client\_id = "esp32-client-";

client\_id += String(WiFi.macAddress());

Serial.printf("The client %s connects to the public MQTT broker\n", client\_id.c\_str());

if (client.connect(client\_id.c\_str(), mqtt\_username, mqtt\_password)) {

Serial.println("Public EMQX MQTT broker connected");

} else {

Serial.print("failed with state ");

Serial.print(client.state());

delay(2000);

}

}

// Publish and subscribe

client.subscribe(topic1);

client.subscribe(topic2);

client.subscribe(topic3);

client.subscribe(topic4);

client.subscribe(topic5);

}

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

Serial.print("Message arrived in topic: ");

Serial.println(topic);

Serial.print("Message: ");

for (int i = 0; i < length; i++) {

Serial.print((char) payload[i]);

}

String receivedValueStr = ""; // String to accumulate characters

int receivedValue = 0; // Integer to store the final value

for (int i = 0; i < length; i++) {

receivedValueStr += (char)payload[i];

}

// Convert the accumulated string to an integer

receivedValue = atoi(receivedValueStr.c\_str());

Serial.println();

Serial.print("Received value as integer: ");

Serial.println(receivedValue);

Serial.println("-----------------------");

// Send the received character to ATmega328P through UART

if (length > 0) {

switch (receivedValue) {

case 1:

Serial.println("Sending '1' to ATmega328P");

Serial2.write(1);

fanStatus=1;

break;

case 2:

Serial.println("Sending '0' to ATmega328P");

Serial2.write(2);

fanStatus=0;

break;

case 3:

Serial.println("Turning Light 1 on");

Serial2.write(3);

LED1Status = 1;

break;

case 4:

Serial.println("Turning Light 1 off");

Serial2.write(4);

LED1Status = 0;

break;

case 5:

Serial.println("Turning Light 2 on");

Serial2.write(5);

LED2Status = 1;

break;

case 6:

Serial.println("Turning Light 2 off");

Serial2.write(6);

LED2Status = 0;

break;

case 7:

Serial.println("Turning Light 3 on");

Serial2.write(7);

LED3Status = 1;

break;

case 8:

Serial.println("Turning Light 3 off");

Serial2.write(8);

LED3Status = 0;

break;

default:

Serial2.write(receivedValue);

fanSpeed = receivedValue;

Serial.println("donee");

break;

}

}

}

void loop() {

client.loop();

delay(1000);

if(count>=10){

if(Serial2.available()){

Serial.print("val received by Atmega: ");

receivedValue = Serial2.read();

Serial.println(receivedValue);

ldrValue = receivedValue;

}

postRequest();

updateValue();

count = 0;

}

count++;

}

void updateValue(){

char payload[200]; // Adjust the size based on the maximum size of your value

snprintf(payload, sizeof(payload), "%d", ldrValue);

// Publish the payload to the MQTT topic

client.publish(topic6, payload);

}

void postRequest(){

// Create the URL for ThingSpeak API

String url = "http://api.thingspeak.com/update?api\_key=" + String(apiKey) +

"&field1=" + String(LED1Status) +

"&field2=" + String(LED2Status) +

"&field3=" + String(LED3Status) +

"&field4=" + String(fanStatus) +

"&field5=" + String(fanSpeed) +

"&field6=" + String(ldrValue);

// Send HTTP POST request

HTTPClient http;

http.begin(url);

int httpResponseCode = http.POST("");

if (httpResponseCode > 0) {

Serial.print("HTTP Response code: ");

Serial.println(httpResponseCode);

} else {

Serial.print("HTTP POST request failed. Error code: ");

Serial.println(httpResponseCode);

}

http.end();

}

# **Code Documentation**

## **AVR Include Files**

**delay.inc:** This File imports a delay macro.

**UART\_Macros.inc:** This file imports UART\_Related macros.

### **Macros Used**

* + Serial\_read: this macro reads a single byte from the UART buffer and stores in a register.
  + Serial\_begin: this macro initializes the UART protocol to send data to or receive data from UART.
  + Serial\_writeReg: this macro writes a register value to UART.
  + Serial\_writeChar: this macro writes a single character to UART.

### **Labels**

* + RESET\_handler: this label initializes the stack pointer.
  + UART\_RX\_ISR: this label contains code to execute when UART RX complete interrupt is fired. Used Interrupts to handle parallel processing.
  + UART\_RX\_ISR\_End: this label returns from the interrupt to where it was interrupted from.
  + fan2\_speed: this label changes the speed of fan 2 by setting its PWM Duty Cycle in Timer 0A register.
  + turn\_light1\_on: this label turns light 1 on.
  + turn\_light1\_off: this label turns light 1 off.
  + turn\_light2\_on: this label turns light 2 on.
  + turn\_light2\_off: this label turns light 2 off.
  + turn\_light3\_on: this label turns light 3 on.
  + turn\_light3\_off: this label turns light 3 off.
  + turn\_fan1\_on: this label turns fan 1 on.
  + turn\_fan1\_off: this label turns fan 1 off.
  + loop: this label takes the ldr values in a register.

### **ESP32 Libraries Used**

* + WiFi.h: It provides functions and methods to connect your ESP32 to a Wi-Fi network, allowing the device to communicate over the Internet or local networks.
  + PubSubClient.h: It allows your ESP32 to act as an MQTT client, enabling it to publish messages to topics or subscribe to topics for receiving messages.
  + HTTPClient.h: This library allows you to make HTTP requests from your ESP32. It is useful for fetching data from web servers, making API calls, or sending data to a server.

# **MQTT Dash app’s dashboard screenshots**

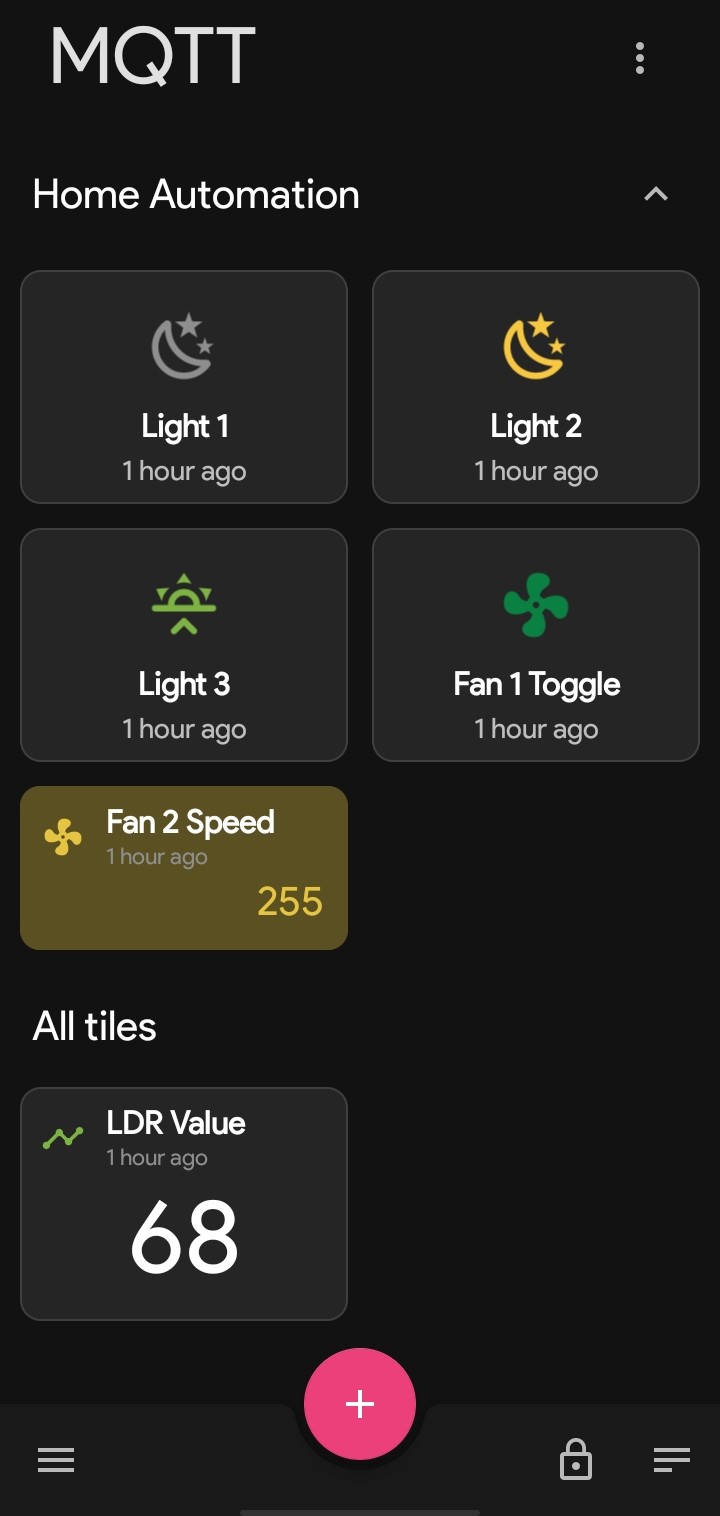


Figure 5:Lights and Fans Control

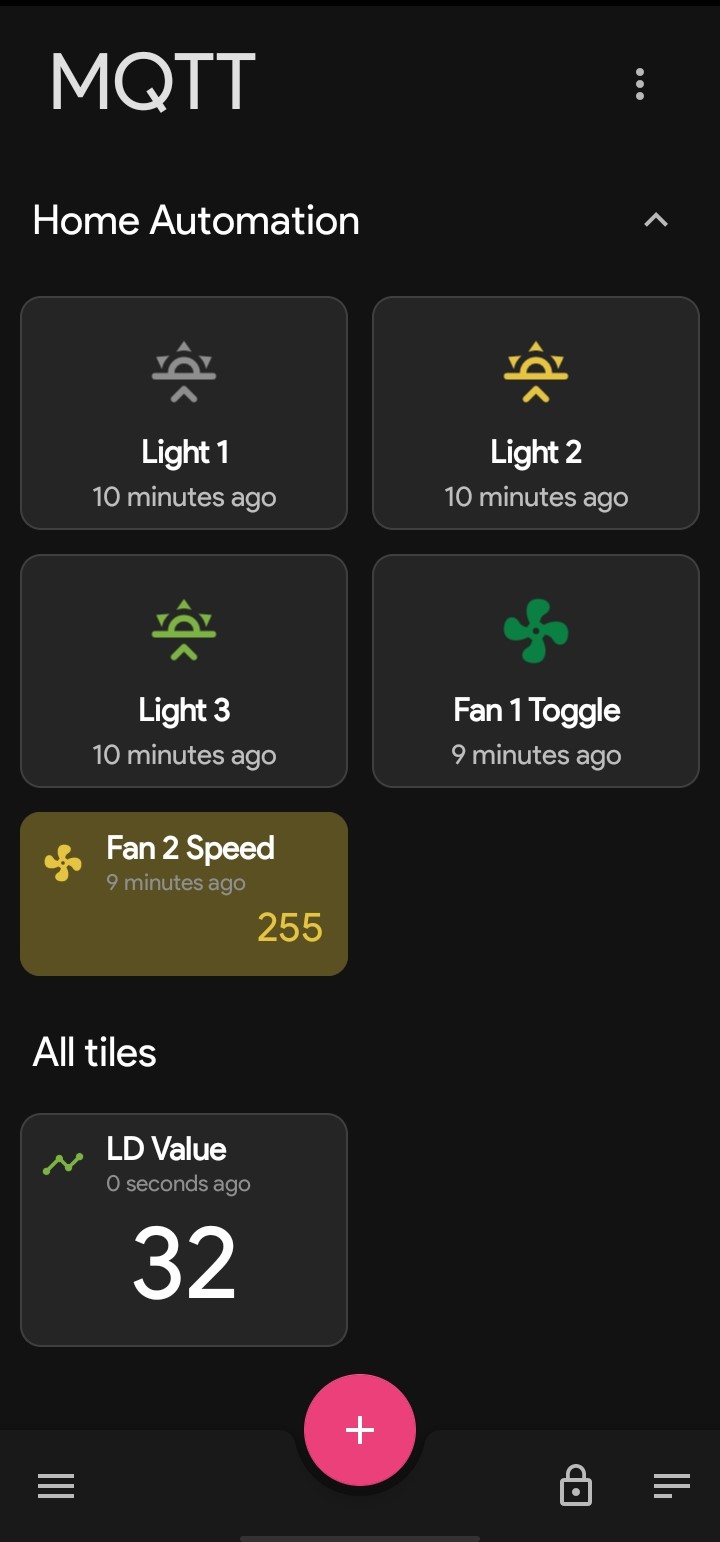


Figure 6: LDR Value

# **Video Links**

<https://www.linkedin.com/posts/activity-7148653567489327104-_P6d?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/activity-7148652105732444160-Q-Oi?utm_source=share&utm_medium=member_desktop>

<https://www.linkedin.com/posts/rafiya-rehan-19645928b_unlocking-the-magic-of-tech-convergence-activity-7148658436086427648-2iex?utm_source=share&utm_medium=member_desktop>

https://www.linkedin.com/posts/ayesha-shafqat-71490b262\_unlocking-the-magic-of-tech-convergence-activity-7148653870955630594-oLuA?utm\_source=share&utm\_medium=member\_desktop

# **GitHub Link**

<https://github.com/AsimAliMurtaza/Home-Automation>

# **References**

<https://www.microchip.com/en-us/product/atmega328p>

<https://www.youtube.com/watch?v=ICIKWlUjYuw>

<https://drive.google.com/file/d/1ExGV801FR6NAzV-AXje-C8jdOYH-WcOt/view>

<https://github.com/TehseenHasan/AVR_Assembly_Example_Codes_for_Atmega328p>