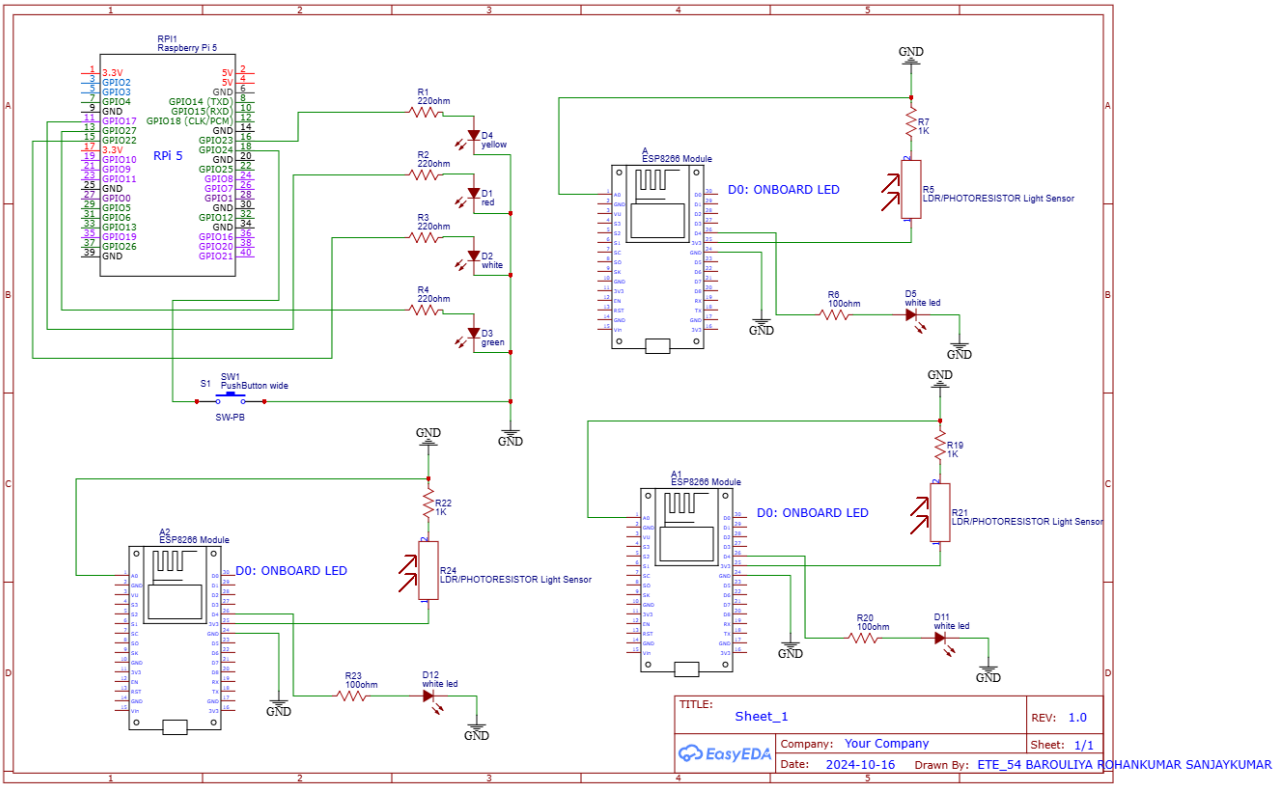
IOT SYSTEMS AND SOFTWARE ASSIGNMENT 4: THE PLOT THICKENS:

SCHEMATIC:



Wiring Connections:

- **Raspberry Pi GPIO Pins:**
  - Pin 17 → Red LED
  - Pin 22 → White LED
  - Pin 27 → Green LED
  - Pin 23 → Yellow LED
  - Pin 24 → Push Button
  - Ground pins connected to each LED ground
- **ESP8266:**
  - Analog Pin (A0) → Photoresistor
  - Pin GPIO0 (D3) → LED Indicator (for brightness)
  - Pin GPIO16 (D0) → Master Status Onboard LED

Raspberry Pi Code (using python3):

```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import socket
from threading import Thread
from datetime import datetime
import time
import numpy as np
from itertools import cycle
```

```
from gpiozero import LED, Button
# GPIO Setup
red_led = LED(17)
green_led = LED(27)
white_led = LED(22)
yellow_led = LED(23)
button = Button(24, pull_up=True)

# Data Storage
device_master_time = {}
sensor_data = []
timestamps = []
device_colors = {}  # Maps device IPs to colors
master_change_points = []  # Records when the master changes
color_cycle = cycle(["blue", "green", "red", "orange", "purple", "cyan", "magenta"])
current_master = None
last_master_update = time.time()

# LED Color Mapping for Master Devices
led_mapping = {
   "master_1": red_led,
   "master_2": green_led,
   "master_3": white_led
}
# UDP Setup
UDP_IP = "0.0.0.0"
UDP_PORT = 4210
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT))

# Socket Listener in a Separate Thread
def socket_listener():
   global current_master, last_master_update, master_change_points, device_colors
   while True:
      data, addr = sock.recvfrom(1024)
      print(f"Raw data received: {data} from {addr}")

      try:
         light_reading = int(data.decode().strip())
         ip = addr[0]
         timestamp = time.strftime("%H:%M:%S")

         # Update sensor data
         sensor_data.append(light_reading)
         timestamps.append(timestamp)

         # Update current master
         if ip != current_master:
```

```
            if current_master is not None:
                elapsed_time = time.time() - last_master_update
                if current_master not in device_master_time:
                    device_master_time[current_master] = 0
                device_master_time[current_master] += elapsed_time
            current_master = ip
            last_master_update = time.time()
            master_change_points.append(len(sensor_data) - 1)

            # Ensure current master has an entry in device_master_time
            if current_master not in device_master_time:
                device_master_time[current_master] = 0
            # Update LED based on current master (using dynamic color mapping)
            if current_master not in device_colors:
                device_colors[current_master] = next(color_cycle)

            # Reset all LEDs before turning on the LED for the current master
            for led in led_mapping.values():
                led.off()

            # Turn on LED for the current master
            led_color = device_colors[current_master]
            if led_color == "blue":
                led_mapping["master_1"].on()
            elif led_color == "green":
                led_mapping["master_2"].on()
            elif led_color == "red":
                led_mapping["master_3"].on()

        # Initialize master history if new
        if ip not in device_master_time:
            device_master_time[ip] = 0

    except ValueError:
        print(f"Malformed data: {data}")
        continue

# Plot Updates
def update_graph(frame):
    global timestamps, sensor_data, device_master_time, device_colors, current_master,
last_master_update, master_change_points

    ax1.clear()
    ax2.clear()
    # Update Master Duration Accurately
    if current_master:
        elapsed_time = time.time() - last_master_update
        if current_master not in device_master_time:
```

```
            device_master_time[current_master] = 0
        device_master_time[current_master] += elapsed_time
        last_master_update = time.time()

    # Plot the sensor data continuously
    if timestamps and sensor_data:
        # Loop through the sensor data and change colors based on the master change points
        for i in range(1, len(sensor_data)):
            start_idx = i - 1
            end_idx = i

            # Determine the color for the current segment
            if len(master_change_points) > 0 and start_idx >= master_change_points[0]:
                # New master change point, update color
                master_ip = current_master
                color = device_colors.get(master_ip, "orange")
                master_change_points.pop(0)
            else:
                # Use the color of the current master
                color = device_colors.get(current_master, "orange")

            ax1.plot([start_idx, end_idx], [sensor_data[start_idx], sensor_data[end_idx]], color=color)

        ax1.set_title("Photocell Data (Waveform)")
        ax1.set_xlabel("Time Index")
        ax1.set_ylabel("Sensor Value")

    else:
        ax1.text(0.5, 0.5, "No data available", ha="center", va="center", fontsize=12)

    # Plot Master Device Bar Chart
    if device_master_time:
        for ip, time_active in device_master_time.items():
            if ip not in device_colors:
                device_colors[ip] = next(color_cycle)
            ax2.bar(ip, time_active, color=device_colors[ip], label=f"{ip} ({time_active:.1f}s)")
        ax2.set_title("Master Device Duration")
        ax2.set_xlabel("Device IP")
        ax2.set_ylabel("Time as Master (s)")
        ax2.legend()
    else:
        ax2.text(0.5, 0.5, "No master data available", ha="center", va="center", fontsize=12)
# Button Handler
def handle_button_press():
    global device_master_time, sensor_data, timestamps, master_change_points
    print("Button pressed! Resetting ESP8266 devices.")
    yellow_led.on()
    time.sleep(3)
```

```
    yellow_led.off()

    # Save current log file
    log_filename = f"master_logs_{datetime.now().strftime('%Y-%m-%d_%H-%M-%S')}.txt"
    with open(log_filename, "w") as log_file:
        log_file.write(f"Master History Log ({log_filename}):\n")
        for ip, time_active in device_master_time.items():
            log_file.write(f"Device {ip}: {time_active:.1f}s\n")
        log_file.write("\nRaw Sensor Data:\n")
        for timestamp, data in zip(timestamps, sensor_data):
            log_file.write(f"{timestamp} - {data}\n")
    print(f"Logs saved to {log_filename}")

    # Reset data for new log
    device_master_time.clear()
    sensor_data.clear()
    timestamps.clear()
    master_change_points.clear()

button.when_pressed = handle_button_press

# Start Socket Listener
thread = Thread(target=socket_listener, daemon=True)
thread.start()

# Create Graph
fig, (ax1, ax2) = plt.subplots(2, 1)
plt.subplots_adjust(hspace=0.5)

ani = FuncAnimation(fig, update_graph, interval=1000)

# Show Graph
try:
    plt.show()
except KeyboardInterrupt:
    print("Program interrupted.")
finally:
    sock.close()
```

ESP8266 Code:

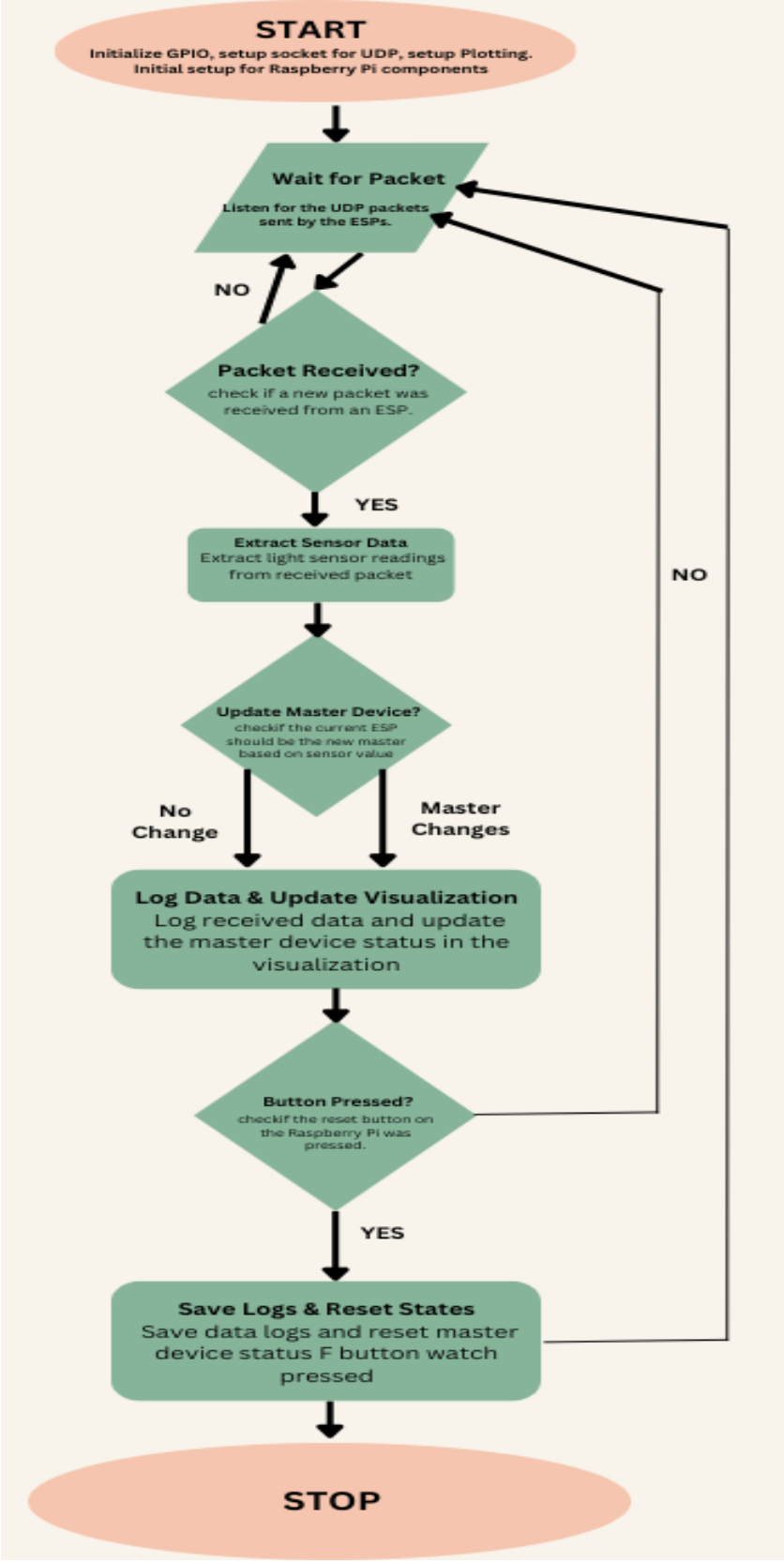
```
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>
#include <WiFiUdp.h>

// Pin Definitions
#define LIGHT_SENSOR_PIN A0      // Analog light sensor
#define LED_INDICATOR 0          // External LED to indicate brightness
#define MASTER_INDICATOR 2       // Onboard LED to indicate master status (active-low)
```

```
// Network and Communication Settings
#define BROADCAST_PORT 4210
#define UDP_PACKET_SIZE 50

ESP8266WiFiMulti WiFiMulti;
WiFiUDP udp;

String broadcastIP = "255.255.255.255"; // Broadcast IP for ESP communication
int lightReading = 0;
bool isMaster = false;
unsigned long lastBroadcastTime = 0;
const int broadcastInterval = 100; // in ms

void setup() {
  Serial.begin(115200);

  // WiFi Connection Setup
  WiFiMulti.addAP("rohan", "12345678"); // Replace with your network credentials
  while (WiFiMulti.run() != WL_CONNECTED) {
    delay(100);
    Serial.print(".");
  }
  Serial.println("\nConnected to WiFi");

  // Begin UDP Communication
  udp.begin(BROADCAST_PORT);

  // Pin Mode Setup
  pinMode(LIGHT_SENSOR_PIN, INPUT);
  pinMode(LED_INDICATOR, OUTPUT);
  pinMode(MASTER_INDICATOR, OUTPUT);

  // Ensure Master LED is off initially (active-low logic)
  digitalWrite(MASTER_INDICATOR, HIGH);
}

void loop() {
  // Read Light Sensor Value
  lightReading = analogRead(LIGHT_SENSOR_PIN);

  // Adjust External LED Brightness Using PWM
  int pwmValue = map(lightReading, 0, 1023, 0, 255); // Map sensor readings to PWM range
  analogWrite(LED_INDICATOR, pwmValue);

  // Broadcast Light Reading Periodically
  if (millis() - lastBroadcastTime > broadcastInterval) {
    broadcastLightReading();
```

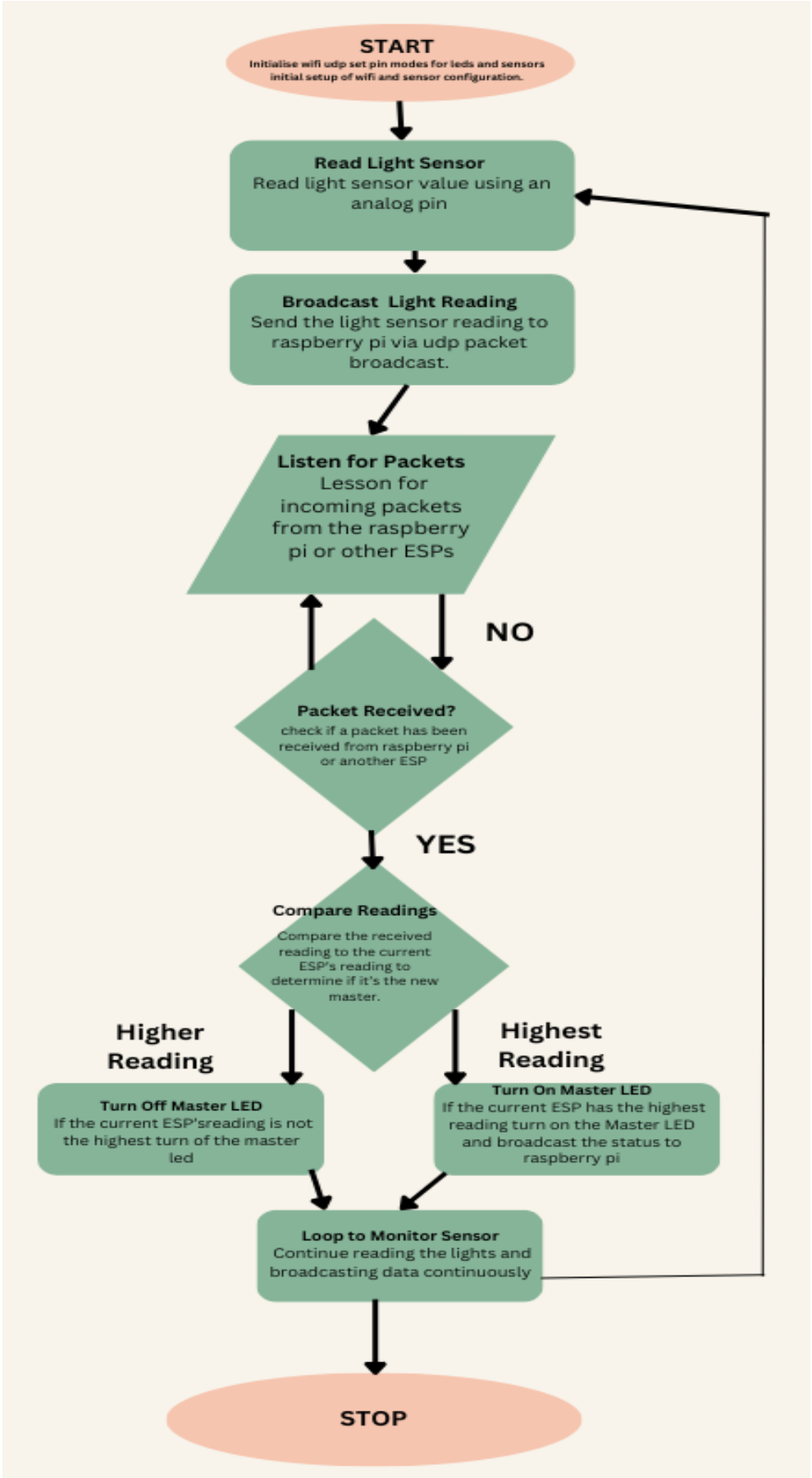
```
    lastBroadcastTime = millis();
    Serial.println("Broadcasting light reading: " + String(lightReading));
  }

  // Listen for Other ESP Broadcasts
  int packetSize = udp.parsePacket();
  if (packetSize) {
    char packetBuffer[UDP_PACKET_SIZE];
    udp.read(packetBuffer, UDP_PACKET_SIZE);
    int otherReading = String(packetBuffer).toInt();
    Serial.println("Received light reading: " + String(otherReading));

    // Determine Master Based on Highest Light Reading
    if (otherReading > lightReading) {
      isMaster = false;
      digitalWrite(MASTER_INDICATOR, HIGH); // Turn Off Master LED if Not Master
      Serial.println("Another ESP has a higher reading. Not the Master.");
    } else {
      isMaster = true;
    }
  }

  // Update Master LED State
  if (isMaster) {
    digitalWrite(MASTER_INDICATOR, LOW); // Turn On Master LED if Master
    sendMasterData();
  }
}

// Function to Broadcast Light Reading to the Swarm
void broadcastLightReading() {
  String reading = String(lightReading);
  udp.beginPacket(broadcastIP.c_str(), BROADCAST_PORT);
  udp.write(reading.c_str());
  udp.endPacket();
}

// Function to Send Data to Raspberry Pi or Log Master's Status
void sendMasterData() {
  // Master Status Information
  Serial.println("==== MASTER STATUS ====");
  Serial.println("Device IP: " + WiFi.localIP().toString());
  Serial.println("Light Reading: " + String(lightReading));
  Serial.println("=======================");
}
```

FLOWCHART:

A) Raspberry Pi:



B) <u>ESP8266:</u>


START
Initialise wifi udp set pin modes for leds and sensors initial setup of wifi and sensor configuration.
Read Light Sensor
Read light sensor value using an analog pin
Broadcast Light Reading
Send the light sensor reading to raspberry pi via udp packet broadcast.
Listen for Packets
Lesson for incoming packets from the raspberry pi or other ESPs
NO
Packet Received?
check if a packet has been received from raspberry pi or another ESP
YES
Compare Readings
Compare the received reading to the current ESP's reading to determine if it's the new master.
Higher Reading
Highest Reading
Turn Off Master LED
If the current ESP'sreading is not the highest turn of the master led
Turn On Master LED
If the current ESP has the highest reading turn on the Master LED and broadcast the status to raspberry pi
Loop to Monitor Sensor
Continue reading the lights and broadcasting data continuously
STOP

Communication Protocol used:

The communication protocol used in this IoT swarm system is UDP (User Datagram Protocol), which is lightweight and ideal for real-time, low-overhead communication. UDP allows each ESP8266 to broadcast its sensor readings to a universal IP address (255.255.255.255), enabling all devices in the swarm to listen and respond without needing exact IP addresses. This broadcast capability simplifies network setup and supports dynamic Master selection, as the ESP8266 with the highest reading becomes the Master and sends data directly to the Raspberry Pi. With minimal delay, UDP provides fast data exchanges crucial for real-time responsiveness, without the need for connection setups or ordered packet delivery. Its connectionless nature also allows devices to join or leave freely, making UDP ideal for this flexible, adaptive IoT system.

Part 1: Raspberry Pi WiFi Setup and Packet Delivery

The Raspberry Pi acts as a central hub for the network, configured to receive UDP packets from the ESP8266 devices. It connects to the same WiFi network as the ESPs to establish communication. The Raspberry Pi listens on a designated UDP port (4210) for incoming packets containing light sensor readings from the ESPs. When a packet is received, it extracts the light reading and determines which ESP currently holds the master status based on the highest light intensity. Using its GPIO pins, the Raspberry Pi controls LEDs (red, green, and white), where each LED corresponds to an ESP acting as the master. Upon detecting a change in the master device, the Raspberry Pi updates the LEDs to reflect the new master status and logs the transition. Additionally, the Raspberry Pi dynamically visualizes the sensor data as a waveform and tracks the cumulative time each ESP spends as the master, displaying this information in a bar chart using Matplotlib. A button connected to the Raspberry Pi enables resetting the system, saving logs, and clearing data for new observations.

Part 2: ESP WiFi Setup and Packet Delivery

Each ESP8266 device connects to the same WiFi network as the Raspberry Pi and broadcasts its light sensor readings periodically using UDP to a common broadcast address (255.255.255.255). The ESP reads light intensity using an analog light sensor, maps the reading to control an external LED's brightness via PWM, and transmits the reading to the network. The ESPs also listen for incoming UDP packets from other ESPs. When a packet is received, the ESP compares the transmitted light reading with its own. If the received reading is higher, the ESP relinquishes its master status by turning off its onboard LED. Conversely, if its reading is the highest, it designates itself as the master and turns on its onboard LED, using active-low logic. This decentralized process enables the ESPs to autonomously decide which device should act as the master. The master ESP periodically broadcasts its status, ensuring that the Raspberry Pi and other ESPs are updated with the latest network state. This mechanism showcases efficient distributed decision-making and communication within the network.

CONFIGURATION SCREENSHOTS:

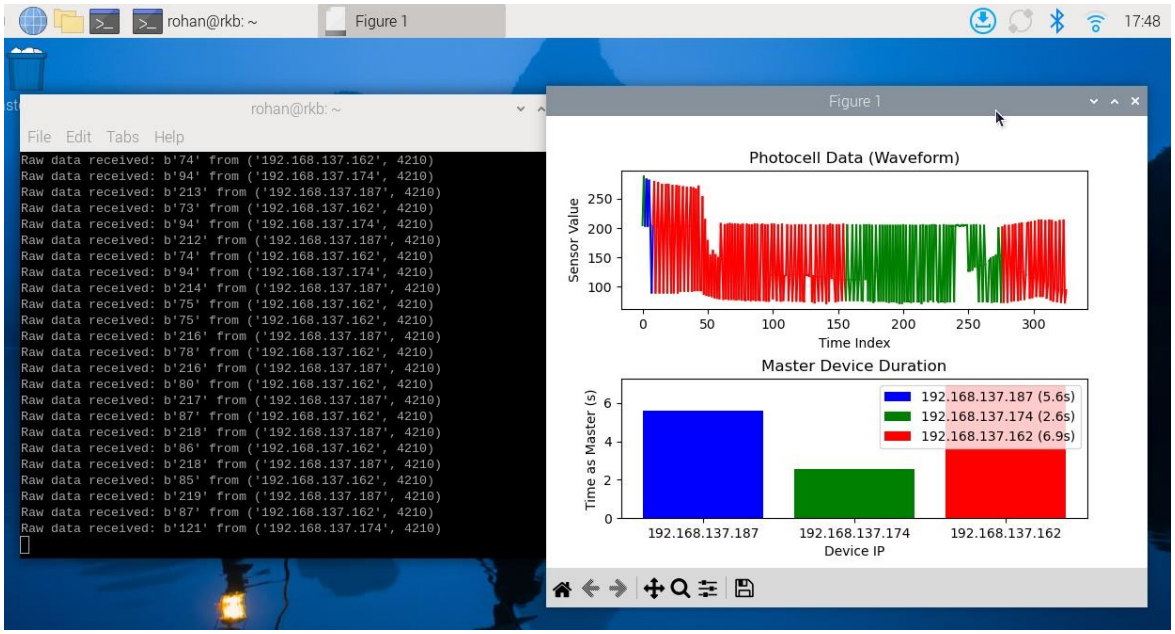
a) Devices connected to network

Name: rohan
Password: 12345678
Band: 2.4 GHz

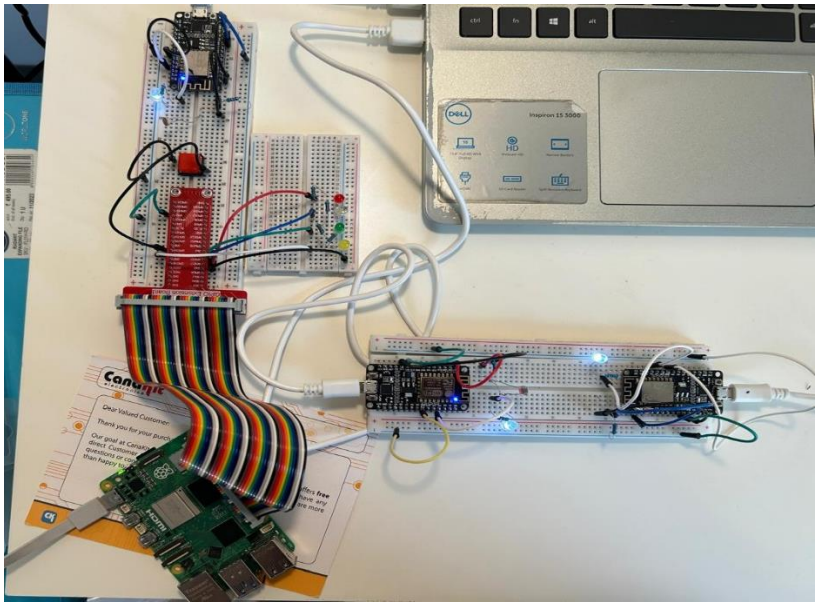
Devices connected: 4 of 8

Device name	IP address	Physical address (MAC)
rkb	192.168.137.201	2c:cf:67:03:0d:98
ESP-D56F96	192.168.137.205	8c:aa:b5:d5:6f:96
ESP-13003D	192.168.137.175	4c:11:ae:13:00:3d
ESP-5F7DE5	192.168.137.160	48:3f:da:5f:7d:e5

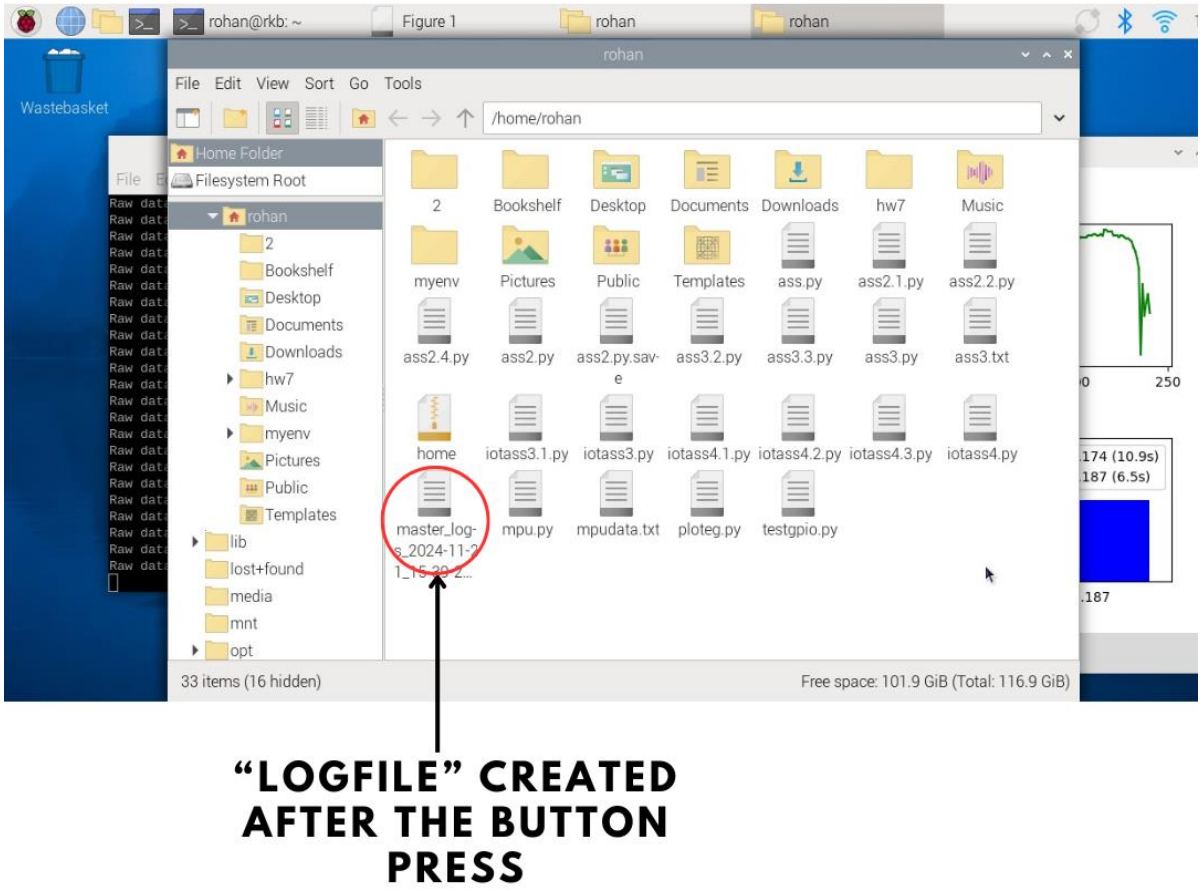
b) Raspberry Pi output visible on terminal along with the photocell data trace & bar chart showing master devices graph.



c) Working Hardware Picture:



d) Logfile being creates after pressing the button, thus getting into reset state, and saving current logfile and running a new logfile to save data from master ESP.



“LOGFILE” CREATED AFTER THE BUTTON PRESS

VIDEO LINK:
https://drive.google.com/file/d/16pz6B1urqqhEdGN8lNN2o_WasZFInLjT/view?usp=sharing