B31OT: Internet of Things

Group 2: Project Report

Duty Cycling and ESP-NOW Communication Between Two ESP32 Sensor Nodes

Group Members:

Vivek Devarapalli (H00512228)
Suhanth Desaraju (H00496045)
12-5-2025

TABLE OF CONTENTS

1. INTRODUCTION

This project presents a low-power wireless sensing system built using two ESP32 microcontrollers operating in a duty-cycled mode and communicating via ESP-NOW. Each node incorporates a DHT11 temperature sensor and an LED indicator (a NeoPixel on Vivek's board and discrete LEDs on Suhanth's board). The system is designed to demonstrate:

- Deep-sleep duty cycling to reduce overall power consumption
- Reliable peer-to-peer ESP-NOW communication between nodes
- Event signalling, where a temperature change detected on one device is reflected on the other
- Energy-efficiency assessment and estimated device lifetime

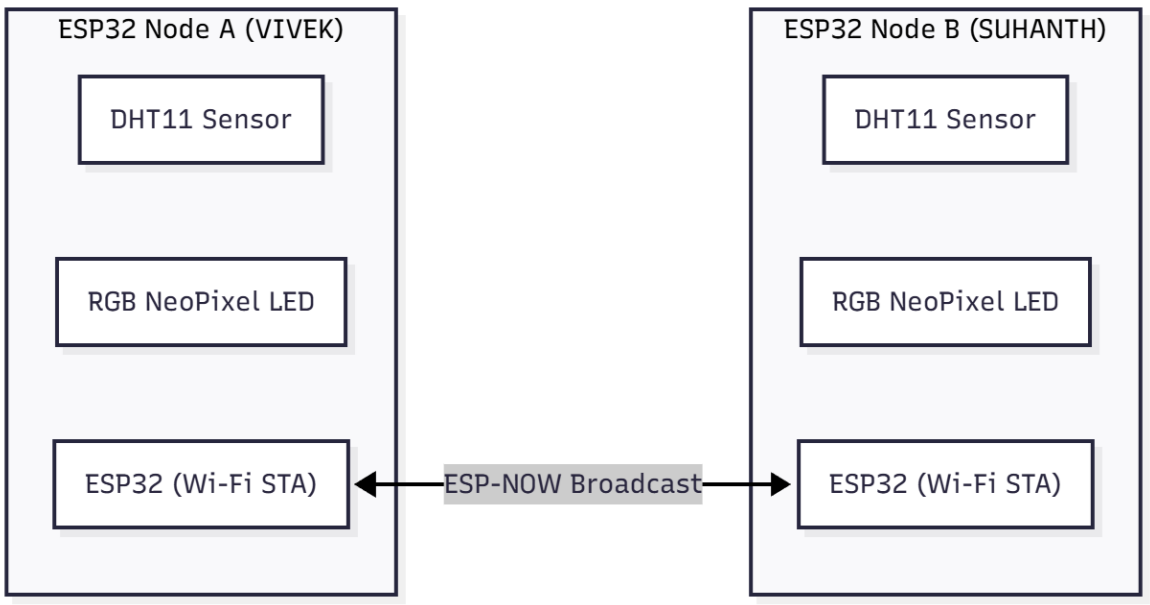
Both ESP32 nodes operate on a periodic schedule consisting of a 10-second awake window followed by 10 seconds of deep sleep. During the awake period, each device samples temperature and humidity, updates its LED state, transmits sensor data every 500 ms, listens for incoming packets, and finally re-enters deep sleep.

A deliberate 1-second phase offset was introduced on Suhanth's device to reduce the chance of simultaneous transmissions and improve the reliability of packet exchange during overlapping active windows.

2. SYSTEM ARCHITECTURE

The system is composed of two independent ESP32 nodes, each performing sensing, local event indication, and peer-to-peer communication. The architecture is symmetric, with both nodes functioning as sensor-transmitter-receivers.

FIGURE 1. PEER-TO-PEER ESP-NOW COMMUNICATION ARCHITECTURE



Each node has the following components:

- **ESP32 (Wi-Fi Station Mode)** - configured on a fixed Wi-Fi channel for ESP-NOW stability
- **DHT11 Temperature Sensor** - measures temperature and humidity
- **LED Indicator** - NeoPixel LED (Vivek) or three indicator LEDs (Suhanth)
- **ESP-NOW Radio Module** - transmits and receives packets containing `temperature`, `humidity`, and `sender ID`

Both nodes broadcast packets using the MAC address `FF:FF:FF:FF:FF:FF`, eliminating the need for explicit peer pairing.

3. IMPLEMENTATION

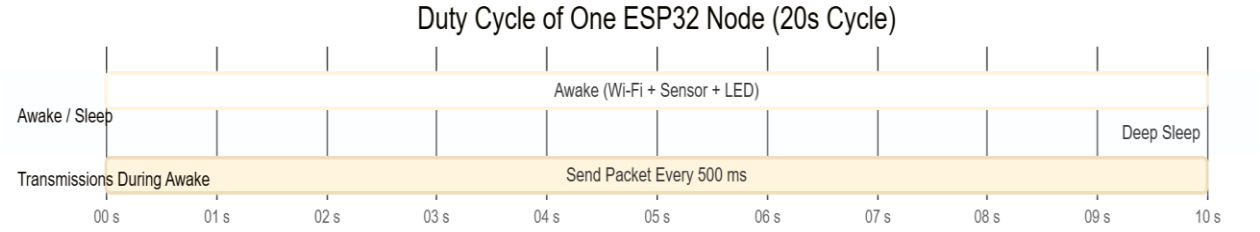
3.1 Duty Cycling Using Deep Sleep

The ESP32's deep sleep mode is used to significantly reduce energy consumption. Each device:

- Wakes every **10 seconds**
- Performs all operations inside the `setup()` function
- Remains awake for approximately **10 seconds**
- Enters **deep sleep** using `esp_deep_sleep_start()`

This resets the microcontroller on each wake, simplifying design (no persistent loop state).

FIGURE 2. DUTY CYCLE TIMING DIAGRAM



During awake time, the following tasks occur:

1. Initialise sensors, ESP-NOW, and LEDs
2. Read DHT11 values
3. Update local LED
4. Start 10-second active window
5. Send ESP-NOW packet every **500 ms**
6. Listen for peer messages and update LED based on remote temperature
7. Log statistics and enter deep sleep

3.2 Justification For 10-Second Awake And Sleep Durations

Shorter awake windows (1-3 seconds) were evaluated but produced unreliable results:

- ESP-NOW packets were often **missed** because awake windows did not overlap reliably
- ESP32 radio requires **hundreds of milliseconds** for stable initialization after waking
- DHT11 requires ~1-2 seconds stabilisation to avoid returning `NaN` values
- Both devices waking simultaneously caused repeated **send-send collisions**
- Receive callbacks often triggered too late in shorter cycles

A **10-second awake window** guarantees that:

- Awake windows overlap between both nodes
- Multiple transmissions (~20 per cycle) increase reliability
- The 1-second offset prevents both devices transmitting simultaneously
- Sensor readings are stable
- Communication is demonstrably reliable for the assignment

Thus, 10s wake / 10s sleep was chosen for correctness and reliability.

3.3 Esp-Now Communication

Both boards were configured with:

```
esp_now_peer_info_t peerInfo = {};
memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.encrypt = false;
esp_now_add_peer(&peerInfo);
```

The message structure:

```
typedef struct {
    char from[10];
    float temp;
    float hum;
} message_t;
```

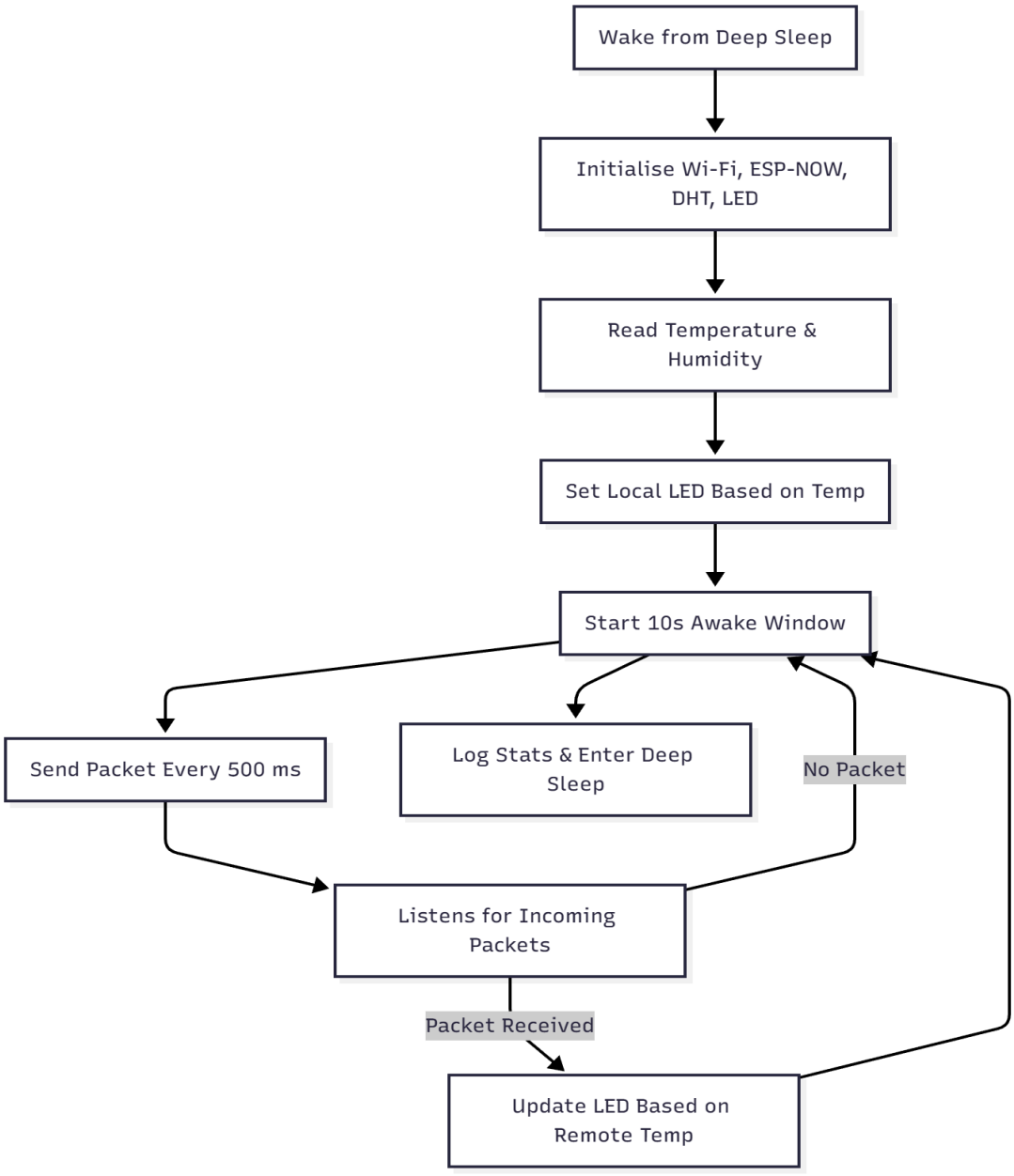
Each device sends:

- Board name (“VIVEK” or “SUHANTH”)
- Temperature and humidity
- Timestamp (embedded in logs)

Incoming packets trigger the receive callback, which:

- Prints MAC address and sender name
- Extracts temperature and humidity
- Updates LED state based on the **remote** device’s temperature
- Increments local receive counter

FIGURE 3. ESP-NOW COMMUNICATION AND PROCESSING FLOW



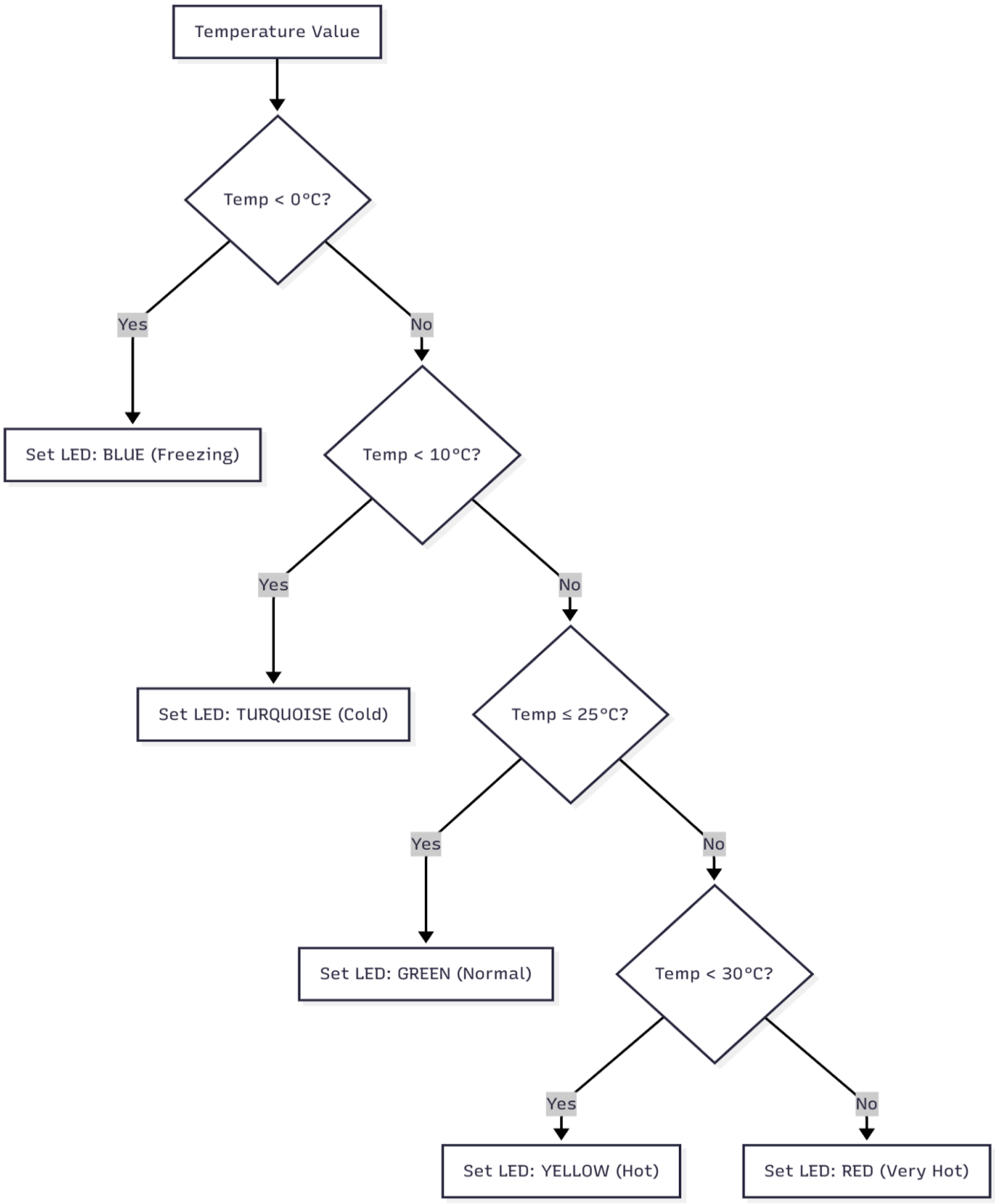
3.4 Temperature Logic And Led Behaviour

Both devices classify temperature into five colour-coded ranges:

Range	LED Colour	Meaning
Temp < 0°C	Blue	Freezing
0-10°C	Turquoise	Cold
10-25°C	Green	Normal
25-30°C	Yellow	Warm
≥ 30°C	Red	Very Hot

Vivek's device uses a NeoPixel LED; Suhanth's device uses GPIO outputs for different LEDs.

FIGURE 4. TEMPERATURE-BASED LED DECISION LOGIC



This logic is applied to **local** readings and **remote** readings, fulfilling the requirement that both nodes indicate temperature changes detected by either one.

4. ENERGY EFFICIENCY AND DEVICE LIFETIME

4.1 Average Current Calculation

Approximate currents:

- **Awake current:** ~80 mA (Wi-Fi + sensor + CPU + LED)
- **Deep sleep current:** ~0.15 mA

The average for a 20-second cycle:

$$I_{avg} = \frac{(80\,mA \times 10\,s + 0.15\,mA \times 10\,s)}{20\,s} = \frac{(800 + 1.5)}{20} \approx 40.08\,mA$$

4.2 Battery Life Estimate

For a **2000 mAh Li-ion cell**:

$$BatteryLife_{hours} = \frac{2000\,mAh}{40.08\,mA} \approx 49.9\,hours$$

$$BatteryLife_{days} = \frac{49.9\,hours}{24} \approx 2.08\,days$$

Duty cycling roughly **halves the consumption** compared to continuous operation (80 mA).

5. RESULTS

Testing produced the following observations:

- **Wake–sleep behaviour:** Both ESP32 nodes consistently woke and entered deep sleep according to the 10-second active and 10-second sleep schedule.
- **Sensor stability:** Temperature readings remained consistent once the DHT11 stabilised after waking, with no invalid values during steady operation.
- **Packet transmission reliability:** Each device transmitted ~20 ESP-NOW packets per active window (one every 500 ms), and the majority were successfully received by the peer.
- **Cross-device LED response:** LED indications on each node correctly reflected both locally measured and remotely received temperature values, confirming proper event propagation.
- **Phase-offset impact:** The 1-second wake offset effectively reduced radio collisions and improved reception rates across successive cycles.
- **Serial output verification:** Logs showed clear cycles of sending, receiving, and entering deep sleep, along with packet counts that aligned with expected behaviour.

Overall, the system demonstrated stable duty-cycled operation, reliable peer-to-peer communication, and accurate event signalling during the evaluation period.

6. DISCUSSION

The system demonstrates stable behaviour across repeated wake–sleep cycles and fulfils the intended functionality. Key observations include:

- **Duty-cycled operation** reduces overall energy consumption by limiting active radio and CPU time while maintaining regular data exchange.
- **ESP-NOW communication** provides fast, lightweight peer-to-peer messaging without Wi-Fi association overhead, making it suitable for short active windows.
- **Phase offset between nodes** improves reliability by reducing simultaneous transmissions and lowering the chance of packet collisions.
- **Temperature-based LED signalling** effectively shows that each device responds not only to its own sensor data but also to readings received from the peer.
- **Energy analysis** confirms that deep-sleep periods significantly decrease average current draw and extend potential operating time.

During development, several challenges were encountered:

- **Sensor stabilisation:** The DHT11 requires a short warm-up period after waking, and reading it too early can produce invalid values.
- **Packet overlap timing:** Reliable ESP-NOW communication depends on sufficient overlap in the active windows of both devices.
- **Synchronisation drift:** Without alignment, both nodes sometimes transmitted simultaneously, reducing receive probability.

These challenges were mitigated by using a longer awake duration, allowing sensor stabilisation time, and adding a **1-second phase offset** between the devices.

7. CONCLUSION & FUTURE SCOPE

This project presented a low-power wireless sensing system using two ESP32 nodes operating under a duty-cycled architecture and communicating through ESP-NOW. The devices successfully exchanged temperature and humidity data during their active windows and updated their LED indicators based on both local and remote readings. The use of deep sleep reduced energy consumption significantly, and the introduction of a small wake-offset between the nodes improved communication reliability by reducing radio collisions. Overall, the system demonstrated stable performance across repeated wake-sleep cycles and consistent peer-to-peer data exchange.

Future Scope:

Several enhancements could further improve the functionality, efficiency, and scalability of the system:

- **Adaptive Duty Cycling:** Implementing dynamic sleep durations based on activity level, rate of environmental change, or battery status could improve energy optimisation.
- **Improved Sensor Performance:** Integrating faster and more precise sensors such as the DHT22 or BME280 would reduce warm-up time and increase measurement accuracy.
- **Time Synchronisation:** Introducing lightweight synchronisation methods (e.g., periodic beacons) could enable shorter awake windows while maintaining reliable packet overlap.
- **Encrypted ESP-NOW Communication:** Enabling encryption would improve security for deployments requiring data confidentiality.
- **Directed Peer Communication:** Configuring ESP-NOW with paired MAC addressing instead of broadcasting could reduce channel noise and improve efficiency.
- **Multi-Node Expansion:** Extending the design to support additional nodes would allow distributed environmental monitoring across larger physical spaces.
- **Cloud or Local Data Logging:** Adding optional integration with MQTT or local storage could enable long-term analysis and remote monitoring when required.

These improvements would allow the system to evolve into a more versatile and scalable IoT platform suitable for a wide range of real-world applications.

8. REFERENCES

[1] Espressif Systems, *ESP32 Technical Reference Manual*, ver. 4.2, 2023. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

[2] Espressif Systems, *ESP-NOW User Guide*, 2023. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/wifi/esp_now.html

[3] Aosong Electronics Co. Ltd., *DHT11 Humidity & Temperature Sensor Datasheet*, 2018. [Online]. Available: https://www.aosong.com/en/products-32.html

[4] Adafruit Industries, *Adafruit NeoPixel Überguide*, 2023. [Online]. Available: https://learn.adafruit.com/adafruit-neopixel-uberguide

[5] R. Santos, "ESP32 Deep Sleep Mode (Arduino IDE)," *Random Nerd Tutorials*, 2023. [Online]. Available: https://randomnerdtutorials.com/esp32-deep-sleep-arduino-ide/

APPENDIX

Appendix A - Source Code

A.1 Vivek's ESP-NOW Node

```
#include <WiFi.h>
#include <esp_now.h>
#include "esp_wifi.h"
#include <Adafruit_NeoPixel.h>
#include <DHTesp.h>

//   Board 1:  #define MY_NAME "VIVEK"
//   Board 2:  #define MY_NAME "SUHANTH"
#define MY_NAME "VIVEK"
// ***********************************************

// ===== NeoPixel =====
#define RGB_PIN     5
#define LED_COUNT   1
#define LED_BRIGHT  100
Adafruit_NeoPixel pixel(LED_COUNT, RGB_PIN, NEO_GRB + NEO_KHZ800);

// ===== DHT Sensor =====
#define DHT_PIN 2
DHTesp dht;

// Deep sleep & awake timings
const uint64_t SLEEP_DURATION_US   = 10ULL * 1000000ULL;  // 10 seconds sleep
const uint32_t AWAKE_DURATION_MS   = 10000UL;            // ~10s awake
const uint32_t SEND_INTERVAL_MS    = 500UL;              // send every 500 ms

// Temperature thresholds (C) – from your MQTT code
const float ALARM_COLD = 0.0;
const float WARN_COLD  = 10.0;
const float WARN_HOT   = 25.0;
const float ALARM_HOT  = 30.0;

// ESP-NOW message structure
typedef struct {
  char  from[10];
  float temp;
  float hum;
} message_t;

message_t outgoingMessage;
message_t incomingMessage;
```

```
// Broadcast MAC address
uint8_t broadcastAddress[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

// RX counter for this wake cycle
volatile uint8_t rxCountThisCycle = 0;

// ---------------- Helpers ----------------
void printMac(const uint8_t* mac) {
  for (int i = 0; i < 6; i++) {
    if (i) Serial.print(":");
    Serial.printf("%02X", mac[i]);
  }
}

void setColor(uint8_t r, uint8_t g, uint8_t b) {
  pixel.setPixelColor(0, pixel.Color(r, g, b));
  pixel.show();
}

void setLedByTemperature(float tempC) {
  uint8_t r = 0, g = 0, b = 0;

  if (tempC < ALARM_COLD) {
    r = 0; g = 0; b = 255;
    Serial.println("LED: BLUE (Freezing)");
  } else if (tempC < WARN_COLD) {
    r = 0; g = 150; b = 150;
    Serial.println("LED: TURQUOISE (Cold)");
  } else if (tempC <= WARN_HOT) {
    r = 0; g = 255; b = 0;
    Serial.println("LED: GREEN (Normal)");
  } else if (tempC < ALARM_HOT) {
    r = 255; g = 200; b = 0;
    Serial.println("LED: YELLOW (Hot)");
  } else {
    r = 255; g = 0; b = 0;
    Serial.println("LED: RED (Very Hot)");
  }

  pixel.setPixelColor(0, pixel.Color(r, g, b));
  pixel.show();
}

// ---------------- Callbacks ----------------

// ESP-NOW Send callback
void onSent(const wifi_tx_info_t *tx_info, esp_now_send_status_t status) {
  Serial.print("[");
  Serial.print(MY_NAME);
  Serial.print("] Send status: ");
```

```
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "SUCCESS" : "FAIL");
}

// ESP-NOW Receive callback
void onReceive(const esp_now_recv_info *recv_info, const uint8_t *data, int len) {
  rxCountThisCycle++;

  Serial.println();
  Serial.println("========= RECEIVED PACKET =========");

  Serial.print("My Name: ");
  Serial.println(MY_NAME);

  Serial.print("From MAC: ");
  printMac(recv_info->src_addr);
  Serial.println();

  if (len == sizeof(message_t)) {
    memcpy(&incomingMessage, data, sizeof(message_t));

    Serial.print("Sender: ");
    Serial.println(incomingMessage.from);

    Serial.print("Temp: ");
    Serial.print(incomingMessage.temp);
    Serial.println(" °C");

    Serial.print("Hum: ");
    Serial.print(incomingMessage.hum);
    Serial.println(" %");

    // Update LED based on received temperature (remote temp)
  setLedByTemperature(incomingMessage.temp);
    // TEMP TEST: force a crazy colour on every received packet
/*Serial.println("[VIVEK] RX TEST -> setting NeoPixel to MAGENTA");
pixel.setPixelColor(0, pixel.Color(0, 0, 255));
pixel.show();*/
  } else {
    Serial.print("Invalid packet size: ");
    Serial.println(len);
  }

  Serial.println("====================================");
  Serial.println();
}

// ---------------- Setup ----------------

void setup() {
  Serial.begin(115200);
  delay(300);
```

```
Serial.println("\n====================================");
Serial.print("Booting ");
Serial.println(MY_NAME);
Serial.println("====================================");

// Init NeoPixel
pixel.begin();
pixel.setBrightness(LED_BRIGHT);
setColor(50, 50, 50); // dim grey at boot

// Init DHT sensor (like in your MQTT code)
dht.setup(DHT_PIN, DHTesp::DHT11);
delay(1500); // allow sensor to stabilise

// Configure WiFi (station mode, fixed channel)
WiFi.mode(WIFI_STA);
WiFi.disconnect();
esp_wifi_set_promiscuous(false);
esp_wifi_set_channel(1, WIFI_SECOND_CHAN_NONE);

uint8_t ch;
wifi_second_chan_t sch;
esp_wifi_get_channel(&ch, &sch);
Serial.print("WiFi channel: ");
Serial.println(ch);

Serial.print("My MAC: ");
Serial.println(WiFi.macAddress());

// Init ESP-NOW
if (esp_now_init() != ESP_OK) {
  Serial.println("ESP-NOW init failed!");
  return;
}

esp_now_register_send_cb(onSent);
esp_now_register_recv_cb(onReceive);

// Add broadcast peer
esp_now_peer_info_t peerInfo = {};
memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.channel = ch;
peerInfo.encrypt = false;
esp_now_add_peer(&peerInfo);

Serial.println("[INFO] ESP-NOW Ready");
Serial.println();

// Small phase offset so both boards are not perfectly in sync
if (String(MY_NAME) == "SUHANTH") {
```

```
  Serial.println("[SUHANTH] Applying 1s phase offset before sends...");
  delay(1000);
}

// ---------- Real DHT reading ----------
TempAndHumidity dhtData = dht.getTempAndHumidity();
float temp = dhtData.temperature;
float hum  = dhtData.humidity;

if (isnan(temp) || isnan(hum)) {
  Serial.println("DHT read FAILED (NaN) – keeping LED as boot colour.");
  // You can optionally early-return here or use previous values.
} else {
  Serial.print("Temp: ");
  Serial.print(temp);
  Serial.print(" °C, Humidity: ");
  Serial.print(hum);
  Serial.println(" %");

  // Local LED based on own temp (same as MQTT behaviour)
  setLedByTemperature(temp);

  // Prepare message
  strncpy(outgoingMessage.from, MY_NAME, sizeof(outgoingMessage.from) - 1);
  outgoingMessage.from[sizeof(outgoingMessage.from) - 1] = '\0';
  outgoingMessage.temp = temp;
  outgoingMessage.hum  = hum;
}

// ---------- Awake window: send multiple packets & listen ----------
rxCountThisCycle = 0;
unsigned long start = millis();
unsigned long lastSend = 0;
uint8_t sendCount = 0;

Serial.println("Staying awake for ~10 seconds, sending every 500 ms...");

while (millis() - start < AWAKE_DURATION_MS) {
  unsigned long now = millis();

  if (now - lastSend >= SEND_INTERVAL_MS) {
    lastSend = now;
    sendCount++;

    esp_err_t result = esp_now_send(broadcastAddress,(uint8_t *)&outgoingMessage,sizeof(outgoingMessage));

    Serial.print("[");
    Serial.print(MY_NAME);
    Serial.print("] Send #");
    Serial.print(sendCount);
    Serial.print(" -> ");
```

```
      Serial.println(result == ESP_OK ? "OK" : "ERROR");
    }

    // Allow ESP-NOW/WiFi background tasks to run so RX callbacks fire
    delay(10);
  }

  Serial.print("Awake window done, total sends: ");
  Serial.println(sendCount);
  Serial.print("Packets received this cycle: ");
  Serial.println(rxCountThisCycle);

  Serial.println("Entering deep sleep for 10 seconds...");
  Serial.println("====================================");

  // ---------- Enter deep sleep ----------
  esp_sleep_enable_timer_wakeup(SLEEP_DURATION_US);
  esp_deep_sleep_start();
}

void loop() {

}
```

A.2 Suhanth's ESP-NOW Node (with 1-second offset)

```
#include <WiFi.h>
#include <esp_now.h>
#include "esp_wifi.h"
#include <Adafruit_NeoPixel.h>
#include <DHTesp.h>

//   Board 1:  #define MY_NAME "VIVEK"
//   Board 2:  #define MY_NAME "SUHANTH"
#define MY_NAME "SUHANTH"
// ************************************************

// ===== NeoPixel =====
#define RGB_PIN     5
#define LED_COUNT   1
#define LED_BRIGHT  100
Adafruit_NeoPixel pixel(LED_COUNT, RGB_PIN, NEO_GRB + NEO_KHZ800);

// ===== DHT Sensor =====
#define DHT_PIN 2
DHTesp dht;

// Deep sleep & awake timings
```

```
const uint64_t SLEEP_DURATION_US   = 10ULL * 1000000ULL;  // 10 seconds sleep
const uint32_t AWAKE_DURATION_MS   = 10000UL;             // ~10s awake
const uint32_t SEND_INTERVAL_MS    = 500UL;               // send every 500 ms

// Temperature thresholds (C) – from your MQTT code
const float ALARM_COLD = 0.0;
const float WARN_COLD  = 10.0;
const float WARN_HOT   = 25.0;
const float ALARM_HOT  = 30.0;

// ESP-NOW message structure
typedef struct {
  char  from[10];
  float temp;
  float hum;
} message_t;

message_t outgoingMessage;
message_t incomingMessage;

// Broadcast MAC address
uint8_t broadcastAddress[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

// RX counter for this wake cycle
volatile uint8_t rxCountThisCycle = 0;

// ---------------- Helpers ----------------
void printMac(const uint8_t* mac) {
  for (int i = 0; i < 6; i++) {
    if (i) Serial.print(":");
    Serial.printf("%02X", mac[i]);
  }
}

void setColor(uint8_t r, uint8_t g, uint8_t b) {
  pixel.setPixelColor(0, pixel.Color(r, g, b));
  pixel.show();
}

void setLedByTemperature(float tempC) {
  uint8_t r = 0, g = 0, b = 0;

  if (tempC < ALARM_COLD) {
    r = 0; g = 0; b = 255;
    Serial.println("LED: BLUE (Freezing)");
  } else if (tempC < WARN_COLD) {
    r = 0; g = 150; b = 150;
    Serial.println("LED: TURQUOISE (Cold)");
  } else if (tempC <= WARN_HOT) {
    r = 0; g = 255; b = 0;
    Serial.println("LED: GREEN (Normal)");
```

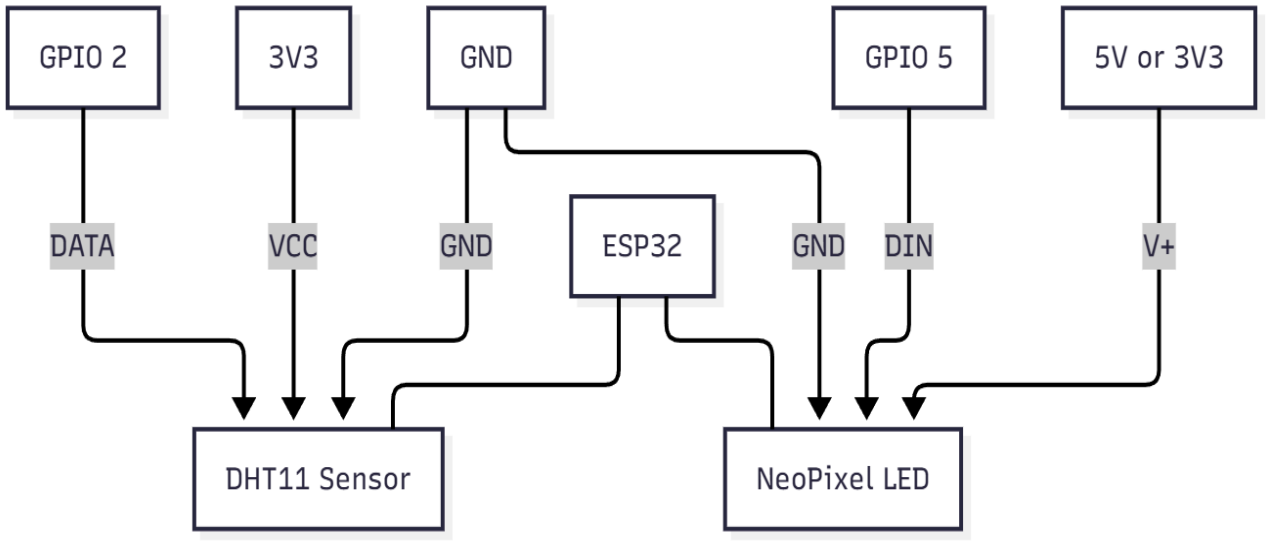
```
  } else if (tempC < ALARM_HOT) {
    r = 255; g = 200; b = 0;
    Serial.println("LED: YELLOW (Hot)");
  } else {
    r = 255; g = 0; b = 0;
    Serial.println("LED: RED (Very Hot)");
  }

  pixel.setPixelColor(0, pixel.Color(r, g, b));
  pixel.show();
}

// ---------- Callbacks ----------

// ESP-NOW Send callback
void onSent(const wifi_tx_info_t *tx_info, esp_now_send_status_t status) {
  Serial.print("[");
  Serial.print(MY_NAME);
  Serial.print("] Send status: ");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "SUCCESS" : "FAIL");
}

// ESP-NOW Receive callback
void onReceive(const esp_now_recv_info *recv_info, const uint8_t *data, int len) {
  rxCountThisCycle++;

  Serial.println();
  Serial.println("========= RECEIVED PACKET =========");

  Serial.print("My Name: ");
  Serial.println(MY_NAME);

  Serial.print("From MAC: ");
  printMac(recv_info->src_addr);
  Serial.println();

  if (len == sizeof(message_t)) {
    memcpy(&incomingMessage, data, sizeof(message_t));

    Serial.print("Sender: ");
    Serial.println(incomingMessage.from);

    Serial.print("Temp: ");
    Serial.print(incomingMessage.temp);
    Serial.println(" °C");

    Serial.print("Hum: ");
    Serial.print(incomingMessage.hum);
    Serial.println(" %");

    // Update LED based on received temperature (remote temp)
```

```
  setLedByTemperature(incomingMessage.temp);
    // TEMP TEST: force a crazy colour on every received packet
/*Serial.println("[VIVEK] RX TEST -> setting NeoPixel to MAGENTA");
pixel.setPixelColor(0, pixel.Color(0, 0, 255));
pixel.show();*/
  } else {
    Serial.print("Invalid packet size: ");
    Serial.println(len);
  }

  Serial.println("=====================================");
  Serial.println();
}

// ---------------- Setup ----------------

void setup() {
  Serial.begin(115200);
  delay(300);

  Serial.println("\n=====================================");
  Serial.print("Booting ");
  Serial.println(MY_NAME);
  Serial.println("=====================================");

  // Init NeoPixel
  pixel.begin();
  pixel.setBrightness(LED_BRIGHT);
  setColor(50, 50, 50); // dim grey at boot

  // Init DHT sensor (like in your MQTT code)
  dht.setup(DHT_PIN, DHTesp::DHT11);
  delay(1500); // allow sensor to stabilise

  // Configure WiFi (station mode, fixed channel)
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  esp_wifi_set_promiscuous(false);
  esp_wifi_set_channel(1, WIFI_SECOND_CHAN_NONE);

  uint8_t ch;
  wifi_second_chan_t sch;
  esp_wifi_get_channel(&ch, &sch);
  Serial.print("WiFi channel: ");
  Serial.println(ch);

  Serial.print("My MAC: ");
  Serial.println(WiFi.macAddress());

  // Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
```

```
  Serial.println("ESP-NOW init failed!");
  return;
}

esp_now_register_send_cb(onSent);
esp_now_register_recv_cb(onReceive);

// Add broadcast peer
esp_now_peer_info_t peerInfo = {};
memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.channel = ch;
peerInfo.encrypt = false;
esp_now_add_peer(&peerInfo);

Serial.println("[INFO] ESP-NOW Ready");
Serial.println();

// Small phase offset so both boards are not perfectly in sync
if (String(MY_NAME) == "SUHANTH") {
  Serial.println("[SUHANTH] Applying 1s phase offset before sends...");
  delay(1000);
}

// ---------- Real DHT reading ----------
TempAndHumidity dhtData = dht.getTempAndHumidity();
float temp = dhtData.temperature;
float hum  = dhtData.humidity;

if (isnan(temp) || isnan(hum)) {
  Serial.println("DHT read FAILED (NaN) – keeping LED as boot colour.");
  // You can optionally early-return here or use previous values.
} else {
  Serial.print("Temp: ");
  Serial.print(temp);
  Serial.print(" °C, Humidity: ");
  Serial.print(hum);
  Serial.println(" %");

  // Local LED based on own temp (same as MQTT behaviour)
  setLedByTemperature(temp);

  // Prepare message
  strncpy(outgoingMessage.from, MY_NAME, sizeof(outgoingMessage.from) - 1);
  outgoingMessage.from[sizeof(outgoingMessage.from) - 1] = '\0';
  outgoingMessage.temp = temp;
  outgoingMessage.hum  = hum;
}

// ---------- Awake window: send multiple packets & listen ----------
rxCountThisCycle = 0;
unsigned long start = millis();
```

```
  unsigned long lastSend = 0;
  uint8_t sendCount = 0;

  Serial.println("Staying awake for ~10 seconds, sending every 500 ms...");

  while (millis() - start < AWAKE_DURATION_MS) {
    unsigned long now = millis();

    if (now - lastSend >= SEND_INTERVAL_MS) {
      lastSend = now;
      sendCount++;

      esp_err_t result = esp_now_send(broadcastAddress,(uint8_t *)&outgoingMessage,sizeof(outgoingMessage));

      Serial.print("[");
      Serial.print(MY_NAME);
      Serial.print("] Send #");
      Serial.print(sendCount);
      Serial.print(" -> ");
      Serial.println(result == ESP_OK ? "OK" : "ERROR");
    }

    // Allow ESP-NOW/WiFi background tasks to run so RX callbacks fire
    delay(10);
  }

  Serial.print("Awake window done, total sends: ");
  Serial.println(sendCount);
  Serial.print("Packets received this cycle: ");
  Serial.println(rxCountThisCycle);

  Serial.println("Entering deep sleep for 10 seconds...");
  Serial.println("=====================================");

  // ---------- Enter deep sleep ----------
  esp_sleep_enable_timer_wakeup(SLEEP_DURATION_US);
  esp_deep_sleep_start();
}

void loop() {

}
```

Appendix B - Wiring Diagram



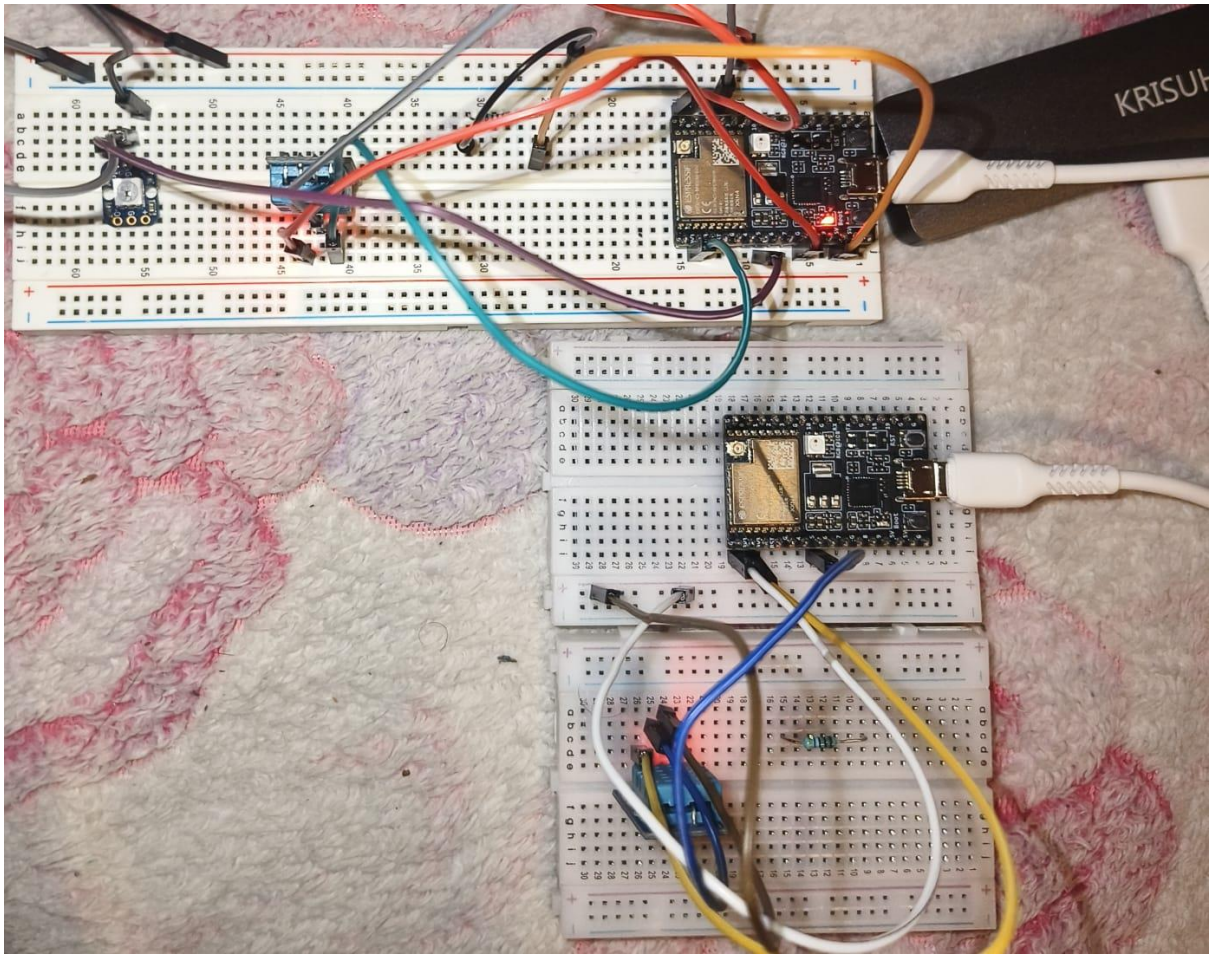
Appendix C - Output Evidence

Photos and screenshots:

- Serial logs showing ESP-NOW send/receive

```
19:14:48.903 -> My Name: SUHANTH
19:14:48.903 -> From MAC: 84:F7:03:12:C7:70
19:14:48.903 -> Sender: VIVEK
19:14:48.903 -> Temp: 20.60 °C
19:14:48.903 -> Hum: 66.00 %
19:14:48.903 -> LED: GREEN (Normal)
19:14:48.903 -> ====================================
19:14:48.903 ->
19:14:49.067 -> [SUHANTH] Send #19 -> OK
19:14:49.067 -> [SUHANTH] Send status: SUCCESS
19:14:49.397 ->
19:14:49.397 -> ========= RECEIVED PACKET =========
19:14:49.397 -> My Name: SUHANTH
19:14:49.397 -> From MAC: 84:F7:03:12:C7:70
19:14:49.397 -> Sender: VIVEK
19:14:49.397 -> Temp: 20.60 °C
19:14:49.397 -> Hum: 66.00 %
19:14:49.397 -> LED: GREEN (Normal)
19:14:49.397 -> ====================================
19:14:49.397 ->
19:14:49.594 -> [SUHANTH] Send #20 -> OK
19:14:49.594 -> [SUHANTH] Send status: SUCCESS
19:14:49.890 ->
19:14:49.890 -> ========= RECEIVED PACKET =========
19:14:49.890 -> My Name: SUHANTH
19:14:49.890 -> From MAC: 84:F7:03:12:C7:70
19:14:49.890 -> Sender: VIVEK
19:14:49.890 -> Temp: 20.60 °C
19:14:49.890 -> Hum: 66.00 %
19:14:49.923 -> LED: GREEN (Normal)
19:14:49.923 -> ====================================
19:14:49.923 ->
19:14:50.089 -> Awake window done, total sends: 20
19:14:50.089 -> Packets received this cycle: 20
19:14:50.089 -> Entering deep sleep for 10 seconds...
19:14:50.089 -> ====================================
```

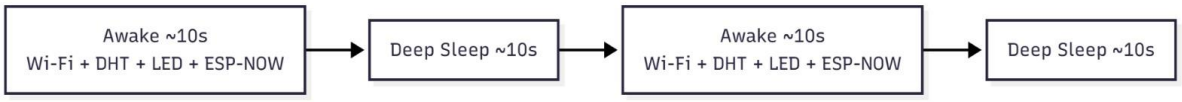
TEMPARATURE VALUES EXCHANGES..

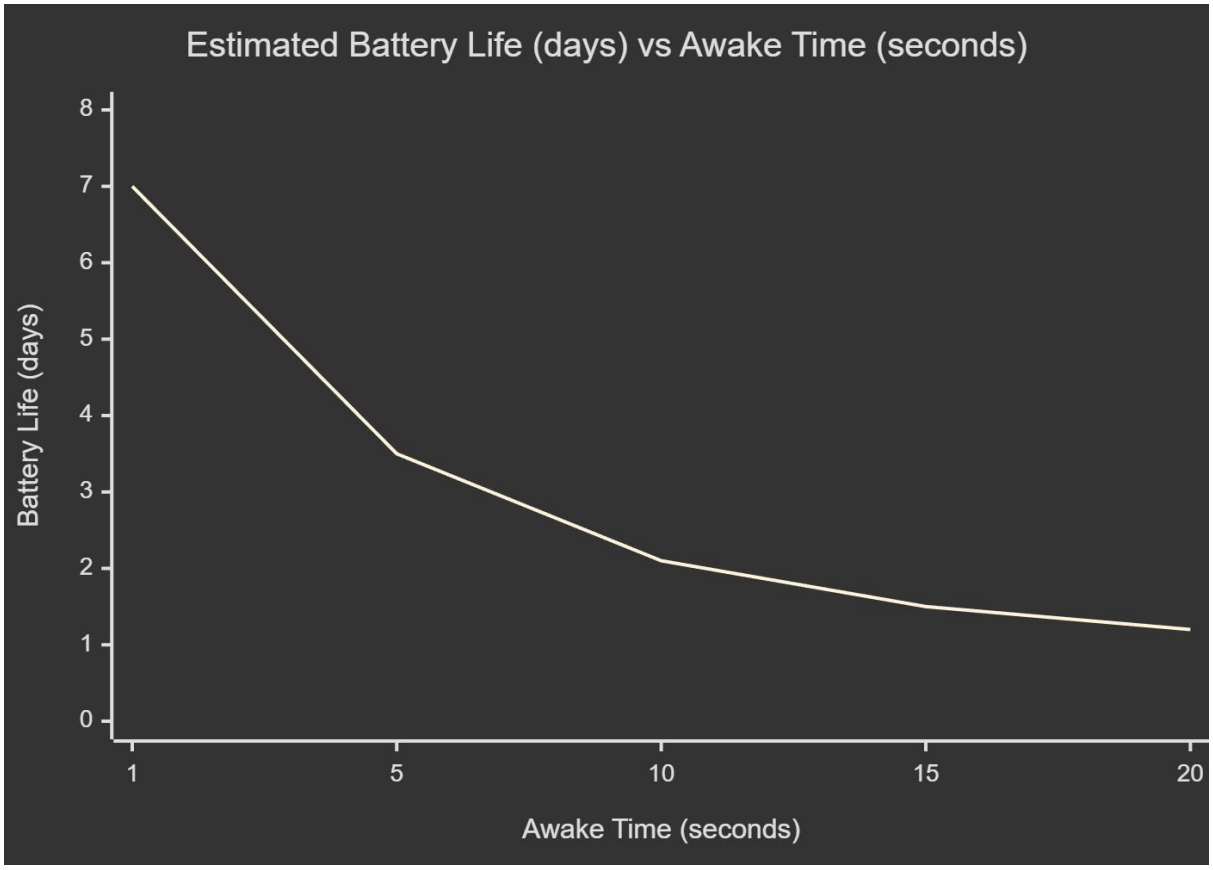


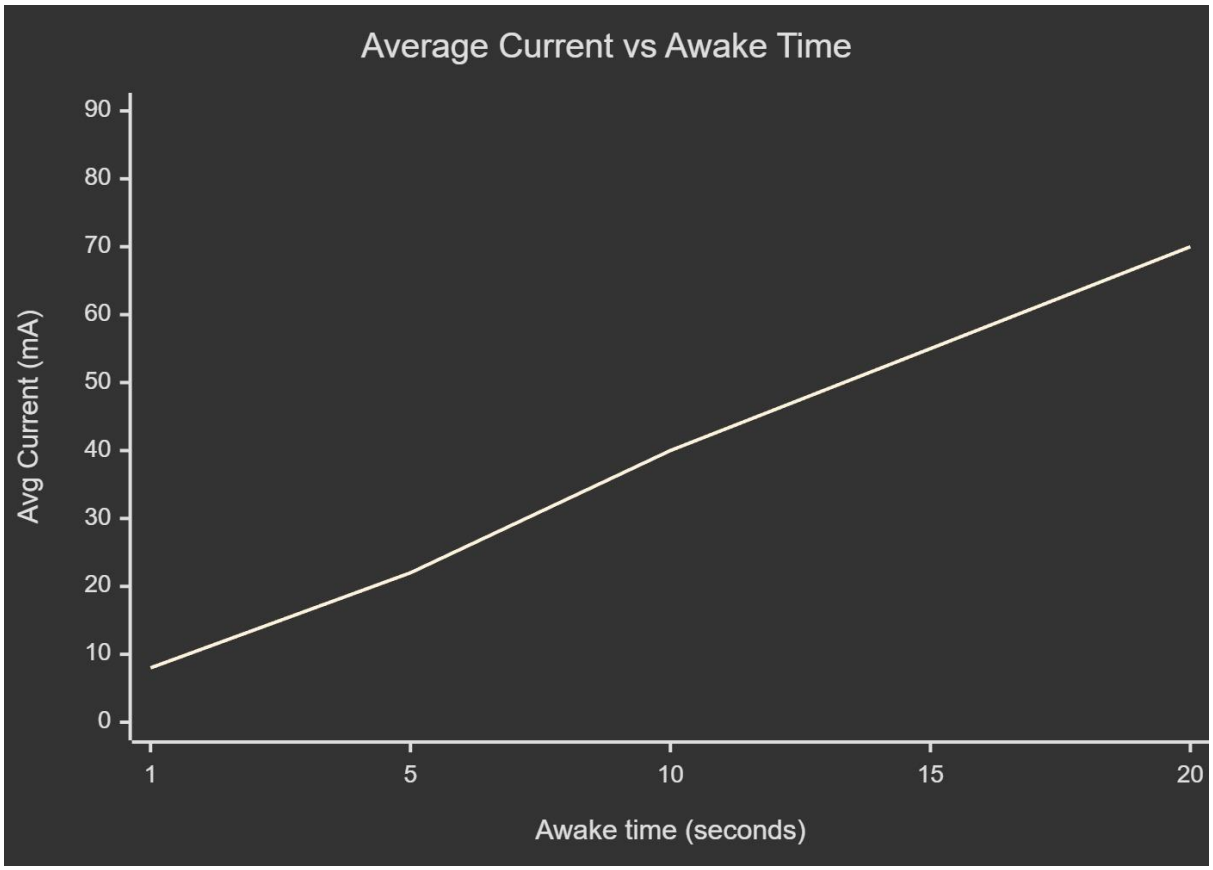
REAL-TIME CONNETION 3

- LED colour behaviour for each category

- Deep sleep timestamps




Estimated Battery Life (days) vs Awake Time (seconds)
Battery Life (days)
8
7
6
5
4
3
2
1
0
1
5
10
15
20
Awake Time (seconds)


Average Current vs Awake Time
Avg Current (mA)
90
80
70
60
50
40
30
20
10
0
1
5
10
15
20
Awake time (seconds)

Appendix D - MAC Addresses

Document MAC addresses printed by each board at startup.