POLITECNICO
MILANO 1863

Internet of Things
Challenge 1

Andrea Pesciotti
10715428

Teachers:
Redondi Alessandro Enrico Cesare
Boiano Antonio

March 2025

1 Parking Occupancy Node Specification

The developed ESP32 node code is organized into functional sections, described below.

1.1 Libraries and Definitions

```
#include <WiFi.h>
#include <esp_now.h>

#define TRIG_PIN 13
#define ECHO_PIN 12

#define uS_TO_S_FACTOR 1000000
#define SLEEP_TIME 33

uint8_t broadcastAddress[] = {0x8C, 0xAA, 0xB5, 0x84, 0xFB, 0x90};
esp_now_peer_info_t peerInfo;
```

1.2 Persistent Variables and Callback

```
RTC_DATA_ATTR int bootCount = 0;

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t
    status) {
     Serial.print("Message status: ");
     Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Success" : "
    Failure");
}
```

1.3 Sensor Reading Function

```
float get_distance() {
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);

    unsigned long duration = pulseIn(ECHO_PIN, HIGH);
    return duration / 58.0;
}
```

1.4 Setup Function

```
void setup() {
    Serial.begin(115200);
    bootCount++;

    pinMode(TRIG_PIN, OUTPUT);
    pinMode(ECHO_PIN, INPUT);

    float distance = get_distance();
    String status = (distance <= 50.0) ? "OCCUPIED" : "FREE";

    WiFi.mode(WIFI_STA);
    if (esp_now_init() != ESP_OK) {
        Serial.println("ESP-NOW initialization failed");
        ESP.restart();
    }

    esp_now_register_send_cb(OnDataSent);
    memcpy(peerInfo.peer_addr, broadcastAddress, 6);
    peerInfo.channel = 0;
    peerInfo.encrypt = false;
    esp_now_add_peer(&peerInfo);

    esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *)
   status.c_str(), status.length() + 1);
    delay(100);

    WiFi.mode(WIFI_OFF);
    esp_sleep_enable_timer_wakeup(SLEEP_TIME * uS_TO_S_FACTOR);
    Serial.flush();
    esp_deep_sleep_start();
}
```

1.5 Main Loop

```
void loop() {
}
```

1.6 WOKWI Simulation

Public project link with commented code:
https://wokwi.com/projects/425667374718896129

---

2 Energy Consumption Estimation

The objective of this section is to estimate the power and energy consumption for a single operational cycle of the sensor node. Additionally, an estimation of the node's lifetime with a battery of given energy capacity will be calculated.

2.1 Power Consumption

The power consumptions for the different operational states were obtained from provided CSV files using Python's `Pandas` library. Unrealistic data points were identified and excluded by applying appropriate thresholds. The resulting values represent the average power consumption for each state.

2.1.1 Idle

The idle state corresponds to the sensor node being active but not performing specific tasks like sensor readings or transmissions. The computed average power consumption for the Idle state, obtained from different measurements, is:

$$\boxed{P_{\text{Idle}} = 321.28\,\text{mW}}$$

2.1.2 Sensor Reading

During this state, the HC-SR04 ultrasonic sensor is active and measuring the distance. The sensor reading power consumption is calculated as:

$$\boxed{P_{\text{Sensor\_Reading}} = 466.74\,\text{mW}}$$

2.1.3 WiFi On

The WiFi on state is characterized by activation of the WiFi module on the ESP32 to establish connectivity. The average power consumption measured in this state is:

$$\boxed{P_{\text{WiFi\_On}} = 775.49\,\text{mW}}$$

2.1.4 Transmission

The transmission state involves sending data packets via ESP-NOW at different transmission power levels. The measured average power consumptions are:

$$\boxed{P_{\text{Tx, 2 dB}} = 797.29\,\text{mW}}$$

$$\boxed{P_{\text{Tx, 19.5 dB}} = 1221.76\,\text{mW}}$$

2.1.5 Deep Sleep

In deep sleep mode, the ESP32 is inactive to minimize energy usage while maintaining only the necessary RTC functions. The average power consumption in deep sleep is:

$$\boxed{P_{\text{Deep\_Sleep}} = 59.66\,\text{mW}}$$

2.2 Energy Consumption

The energy consumption for one transmission cycle has been estimated by measuring the time spent in each state using the `micros()` function in the source code. The measurements are averaged over multiple simulation runs with different sensor reading distances. Although not perfectly precise due to simulation limitations, these values provide a realistic estimation of the timing for each operational phase:

- **Idle Duration:** $1071.59\ \mu s = 1.07159 \times 10^{-3}\ s$
- **Sensor Reading Duration:** $3728.26\ \mu s = 3.72826 \times 10^{-3}\ s$
- **Wi-Fi On Duration:** $191977.03\ \mu s = 0.19197703\ s$
- **Transmission Duration:** $150.88\ \mu s = 1.5088 \times 10^{-4}\ s$
- **Deep Sleep Duration:** $33\ s$

The total energy consumption for a complete transmission cycle is calculated using the following formula:

$$E = (t_{idle} \cdot P_{idle}) + (t_{sensor} \cdot P_{sensor}) + (t_{wifi} \cdot P_{wifi}) + (t_{tx} \cdot P_{tx}) + (t_{sleep} \cdot P_{sleep})$$

Substituting the values (with power converted to Watts):

$$\begin{aligned} E = {} & (1.07159 \times 10^{-3}\ s \cdot 0.32128\ W) \\ & + (3.72826 \times 10^{-3}\ s \cdot 0.46674\ W) \\ & + (0.19197703\ s \cdot 0.77549\ W) \\ & + (1.5088 \times 10^{-4}\ s \cdot 1.22176\ W) \\ & + (33\ s \cdot 0.05966\ W) \end{aligned}$$

Result: Therefore, the total energy consumption of one transmission cycle is approximately:

$$\boxed{2.1200 \text{ Joules}}$$

2.3 Battery Lifetime Estimation

The battery has a total energy capacity of **15,428 Joules**.

Given the following measured durations for each operation in a full transmission cycle:

- **Idle duration:** 1071.59 µs $\rightarrow$ 0.00107159 s
- **Sensor reading duration:** $\frac{2645.44 + 4811.08}{2} = 3728.26$ µs $\rightarrow$ 0.00372826 s
- **Wi-Fi on duration:** 191977.03 µs $\rightarrow$ 0.19197703 s
- **Transmission duration:** 150.88 µs $\rightarrow$ 0.00015088 s
- **Deep sleep duration:** 33 s

The total duration of a full cycle is:

$$\text{Cycle duration} = 0.00107159+0.00372826+0.19197703+0.00015088+33 = 33.1969 \text{ seconds}$$

The energy consumption per full transmission cycle is estimated as:

$$E_{\text{cycle}} = 2.1200 \text{ Joules}$$

The number of complete cycles the node can perform before battery depletion is:

$$\text{Number of cycles} = \frac{15428}{2.1200} \approx 7275 \text{ cycles}$$

Total operating time is:

$$\text{Total duration} = 7275 \times 33.1969 \approx 241493 \text{ seconds}$$

$$\text{Total duration in hours} = \frac{241493}{3600} \approx 67.08 \text{ hours}$$

Result: If the node transmits during every wake-up cycle, the battery will last approximately:

$$\boxed{67 \text{ hours } (\approx \ 3 \text{ days})}$$

---

3 Energy Consumption Improvements

The current implementation of the parking sensor node successfully detects vehicle presence and transmits the occupancy status to the sink node. However, several improvements can further reduce energy consumption and extend the battery life.

3.1 Possible Improvements

- **Wi-Fi Transmission Power Reduction**: The ESP-NOW transmission is currently configured at **19.5 dBm**, resulting in high power consumption (**1221.76 mW**) during transmission. Since the sink node is located at the center of the parking area and the distance is limited, the transmission power can be reduced to **2 dBm**, lowering the transmission power consumption to **797.29 mW** without impacting communication reliability.
- **Deep Sleep Interval Extension**: The current deep sleep duration is set to **33 seconds**. Increasing the sleep interval reduces the duty cycle and significantly lowers average energy consumption, as deep sleep consumes only **59.66 mW**. This is a simple and highly effective way to extend battery life, especially in low-traffic scenarios.
- **Conditional Transmission Based on State Change**: Instead of transmitting the parking status ("FREE" or "OCCUPIED") at every wake-up, the node can store the last detected state in RTC memory. The sensor will only activate Wi-Fi and transmit a message if the parking state has changed compared to the previous cycle. This approach avoids unnecessary Wi-Fi usage and transmission when the parking spot state remains unchanged.

3.2 Wi-Fi Power Reduction

The following plot shows the difference in power consumption when changing the Wi-Fi transmission power:

```
// Wi-Fi setup
WiFi.mode(WIFI_STA);
WiFi.setTxPower(WIFI_POWER_2dBm);
```

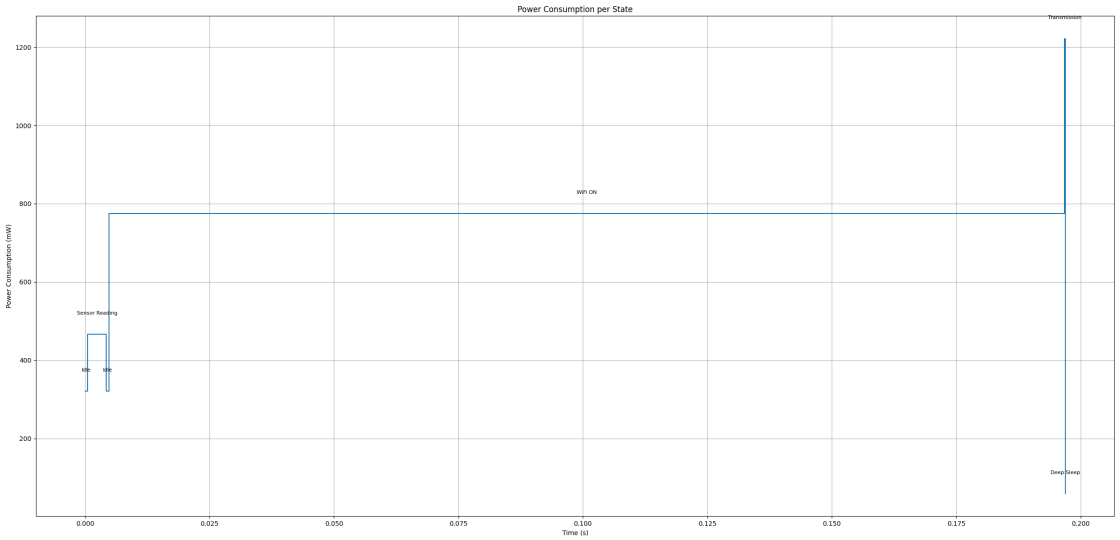

Figure 1: Power consumption profile in the current implementation (19.5 dBm transmission)

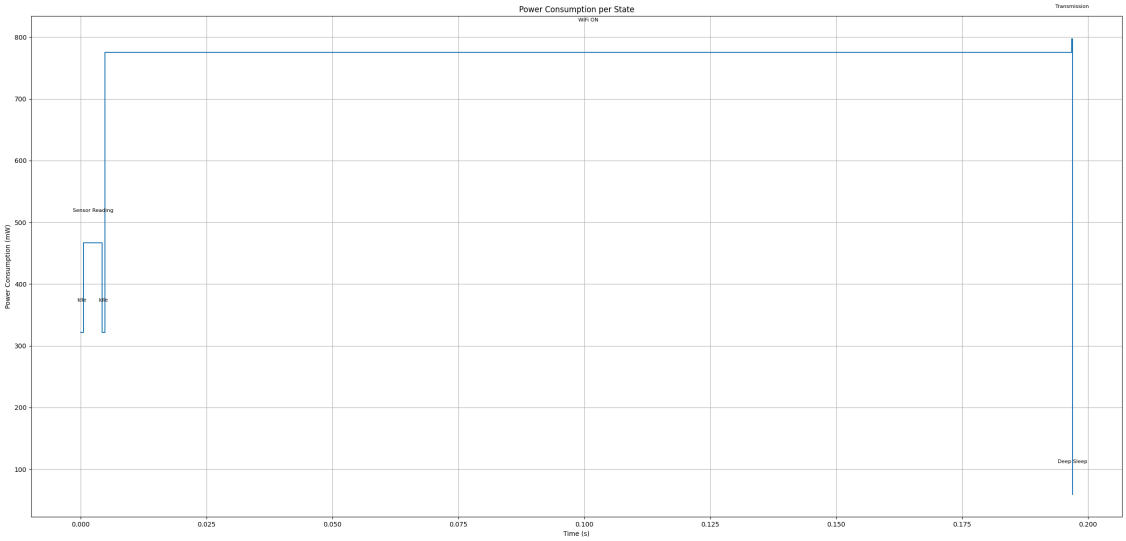

Figure 2: Power consumption profile after reducing Wi-Fi transmission power to 2 dBm

As a result of this change, the total energy consumption for a single transmission cycle decreases from **2.12 Joules** to **2.1199 Joules**. While the reduction may seem small per cycle, it becomes significant over thousands of cycles, directly impacting the battery lifetime and improving the overall energy efficiency of the system.

3.3 Implementation of Conditional Transmission

The following code snippet shows the relevant part to implement the conditional transmission by using `RTC_DATA_ATTR` to store the last known occupancy state:

```
// Store last state in RTC memory
RTC_DATA_ATTR bool lastOccupied = false;

// After distance measurement
bool currentOccupied = (distance <= 50.0);

void setup() {
    // As before

    // Transmit only if the status changes
    if (currentOccupied != lastOccupied) {
        lastOccupied = currentOccupied;  // Store state

        // Wi-Fi and ESP-NOW setup
        WiFi.mode(WIFI_STA);
        WiFi.setTxPower(WIFI_POWER_2dBm);  // Set power to 2 dBm
        if (esp_now_init() != ESP_OK) {
            Serial.println("ESP-NOW init failed");
            ESP.restart();
        }
        esp_now_register_send_cb(OnDataSent);
        memcpy(peerInfo.peer_addr, broadcastAddress, 6);
        peerInfo.channel = 0;
        peerInfo.encrypt = false;
        esp_now_add_peer(&peerInfo);

        // Send the message
        esp_err_t result = esp_now_send(broadcastAddress, (uint8_t
   *)status.c_str(), status.length() + 1);

        // Disable Wi-Fi
        WiFi.mode(WIFI_OFF);
    }

    // Deep sleep
    esp_sleep_enable_timer_wakeup(SLEEP_TIME * uS_TO_S_FACTOR);
    Serial.flush();
    esp_deep_sleep_start();
}
```

Note: Due to `Wokwi`'s simulation limitations, it is not possible to verify the impact of `RTC_DATA_ATTR` persistence during deep sleep. In `Wokwi`, the value is reset at every restart. However, in real hardware, this approach is expected to significantly reduce power consumption by skipping unnecessary Wi-Fi activation and data transmission when no state change is detected.

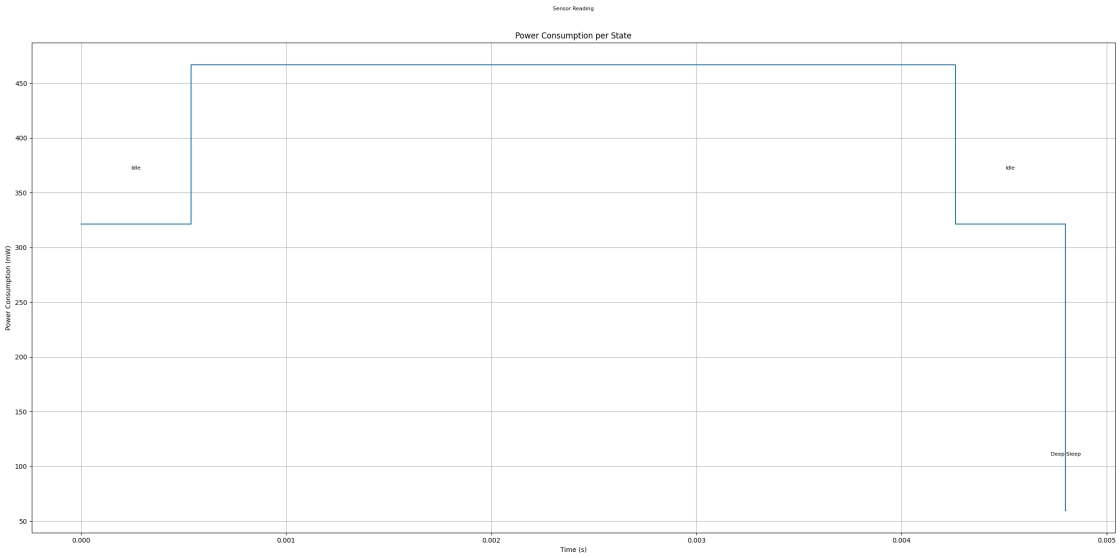

Figure 3: Power consumption profile if no status change