IoT Challenge #1

Parking Occupancy Node and Energy Consumption Analysis

Giuliano Crescimbeni, Arimondo Scrivano
Politecnico di Milano

March 2025

Contents

1 Introduction

The goal of this project is to develop an IoT-based parking occupancy detection node using the ESP32 microcontroller and the HC-SR04 ultrasonic sensor. The node determines whether a parking spot is occupied or free and communicates this information wirelessly using the ESP-NOW protocol. Additionally, an energy consumption analysis is performed to estimate the power usage and battery lifetime.

This document provides an in-depth technical description of the system implementation, including the hardware design, software development, and energy estimation calculations.

2 System Specifications

2.1 Hardware Components

The parking occupancy node consists of the following hardware components:

- **ESP32 Development Board**: Serves as the microcontroller, handling sensor readings, data processing, and communication.
- **HC-SR04 Ultrasonic Sensor**: Measures the distance to detect the presence of a vehicle.

2.2 Working Principle

The system functions as follows:

1. The ESP32 wakes up from deep sleep at predefined intervals (duty cycle X).
2. The HC-SR04 sensor measures the distance to detect the presence of a car.
3. If the distance is <50 cm, the parking spot is considered occupied; otherwise, it is free.
4. The ESP32 sends the occupancy status ("OCCUPIED" or "FREE") to the sink node via ESP-NOW.
5. The ESP32 returns to deep sleep mode.

3 Implementation

3.1 Hardware Implementation

Figure 1 shows the circuit schematic of the system.

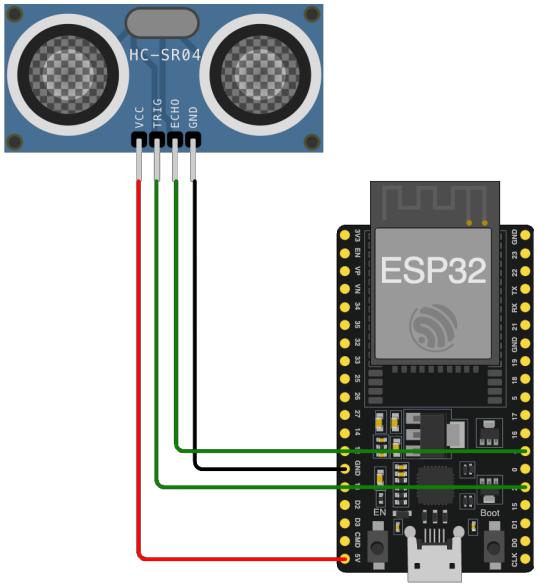

Figure 1: Schematic of the parking occupancy node.

3.2 Software Implementation

3.3 Code Explanation

The main functionalities of the firmware include:

1. Initializing the ESP-NOW communication protocol.
2. Setting up the HC-SR04 sensor and obtaining distance measurements.
3. Checking if a vehicle is present based on the measured distance.
4. Sending the occupancy status as a message to the sink node.
5. Entering deep sleep mode until the next duty cycle.

3.3.1 ESP32 Setup

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

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

    Serial.println("Starting...");
    . . .
```

The **setup()** function starts by initializing the serial communication and configuring the sensor pins.

```
    ...
    // SENSOR READ //
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);

    // Read the result
    int duration = pulseIn(ECHO_PIN, HIGH, 4000);
    unsigned long sensor_end = micros();
    // END OF SENSOR READ //

    // Distance conversion in centimeters
    int distance= duration / 58;

    // Create the message to send
    String snd_msg = (distance < 51 && distance != 0) ?
       "OCCUPIED\0" : "FREE\0";
    ...
```

A short 10-microsecond pulse is sent to the **TRIG_PIN** to trigger the HC-SR04 sensor. The **pulseIn()** function measures the duration of the returning signal on the **ECHO_PIN**, which is then converted to centimeters by dividing by ‘58‘.

The occupancy status is determined based on the measured distance and convertend into a string: - If distance $<$ 51 cm, the spot is occupied. - Otherwise, it is free.

```
    ...
    // WIFI START //
    WiFi.mode(WIFI_STA);

    esp_now_init();
    esp_now_register_send_cb(OnDataSent);

    memcpy(peerInfo.peer_addr,broadcast_addr, 6);
    peerInfo.channel= 0;
    peerInfo.encrypt= 0;

    esp_now_add_peer(&peerInfo);
    ...
```

The ESP WiFi configuration starts now, the ESP-NOW peer is configured by copying the destination address into **peerInfo**, setting the communication channel to **0**, and disabling encryption. The **esp_now_add_peer** function registers the sink node as a valid recipient.

```
    ...
    // TRANSMISSION START //
    esp_now_send(broadcast_addr, (uint8_t*) snd_msg.
       c_str(), snd_msg.length() + 1);
    // TRANSMISSION END //

    WiFi.mode(WIFI_OFF);
    unsigned long wifi_end = micros();
    // WIFI END//

    int personal_duty_cycle= 3+5; // Personal Code =
       107124(03)
    esp_sleep_enable_timer_wakeup(personal_duty_cycle *
       uS_TO_S);
}
```

The information about the occupancy is sent via ESP-NOW to the sink node and WiFi is turned off. The ESP32 is configured to enter deep sleep mode for a predefined duty cycle, calculated based on the personal code. The system will wake up after the defined time interval, repeating the process while minimizing power consumption.

4 Energy Consumption Estimation

4.1 Power Consumption per State

The power consumption value of each state was obtained by analyzing the waveforms of the sample data and taking the mean of the highlighted values like as shown:

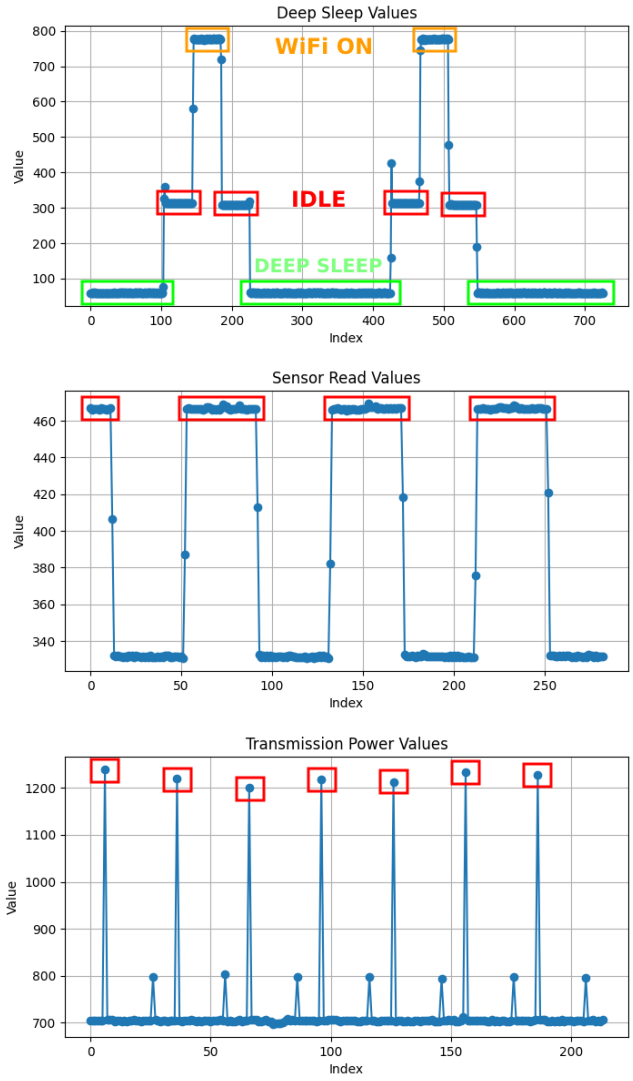

Figure 2: Waveforms for different states: Duty Cycle, Sensor Read, and Transmission Power.

Deep Sleep duration is calculated, as requested, starting from the personal code 10712403 (Giuliano Crescimbeni)

$$\mathbf{03} \bmod 50 + 5 = 8s$$

The sensor read duration, transmission duration, and idle duration have been calculated during the simulation. The transmission period includes WiFi enabling, disabling and the actual transmission.

State	Power Consumption (mW)	Duration (s)
Deep Sleep	1198	8.00
Sensor Read	9320	0.004
Transmission	24780	0.197
Idle	6260	0.190

Table 1: Power consumption for different states.

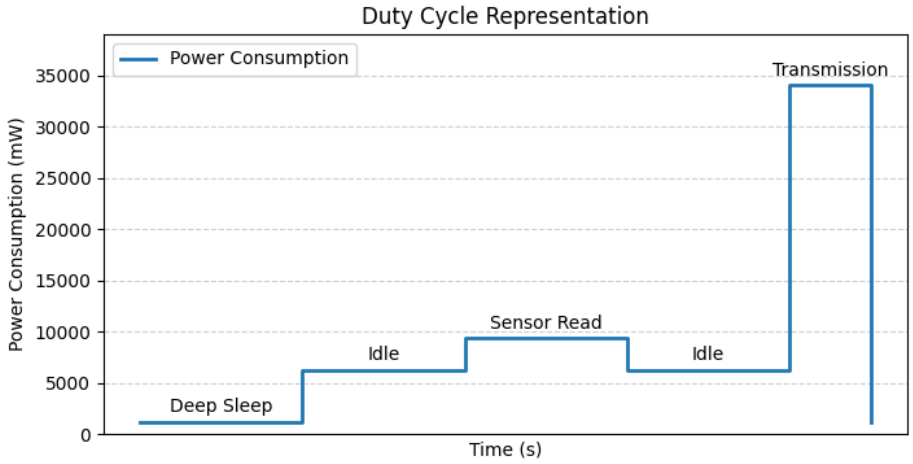

Figure 3: An approximate representation of the duty cycle of the ESP-32.

4.2 Energy Consumption Computation

The energy consumption for each operational state is computed as:

$$E = P_s \times t \tag{1}$$

where:

- E is the energy consumption in millijoules (mJ),
- P is the power consumption in milliwatts (mW),
- t is the duration of the state in seconds (s).

Table 2 reports the energy consumption per state.

State	**Energy Consumption (mJ)**
Deep Sleep	9520.0
Sensor Read	37.77
Transmission	3072.48
Idle	1192.10
Total Energy Consumption	**13.82 J**

Table 2: Energy consumption per cycle.

4.3 Total System Energy and Lifetime Estimation

The total lifetime of the system is estimated as:

$$N_{cycles} = \frac{E_{battery}}{E_{cycle}} \tag{2}$$

$$T_{lifetime} = N_{cycles} \times T_{cycle} \tag{3}$$

where:

- N_{cycles} is the maximum number of duty cycles before battery depletion,
- $E_{battery}$ is the total available energy in joules (J),
- E_{cycle} is the energy consumed per cycle in joules (J),
- T_{cycle} is the duration of one full cycle in seconds (s),
- $T_{lifetime}$ is the total system operation time in seconds (s).

The available energy has been calculated, as requested, from the personal code 10712403 (Giuliano Crescimbeni)

$$\mathbf{2403} \bmod 5000 + 1500 = 17403J$$

The total system lifetime is computed as follows:

- **Max number of cycles:** 1259.04
- **Max period of activity:** 10566.44s (176 minutes)

Key Result: The system is expected to operate for about 193.5 minutes before the battery is depleted.

5 System Optimization and Improvements

To enhance the **lifetime of the system** and reduce overall energy consumption, one of the most effective strategies is to **increase the Deep Sleep duration** of the ESP32 sensor node. By reducing the frequency of wake-ups, the energy spent per unit of time decreases, extending the battery life.

5.1 Deep Sleep Optimization Approach

A simulation was performed by varying the **Deep Sleep interval** to find an optimal trade-off between responsiveness and energy consumption.

In order to find the best possible value, we tried every value in a [8s - 3600s] with a step of 0.5s each:

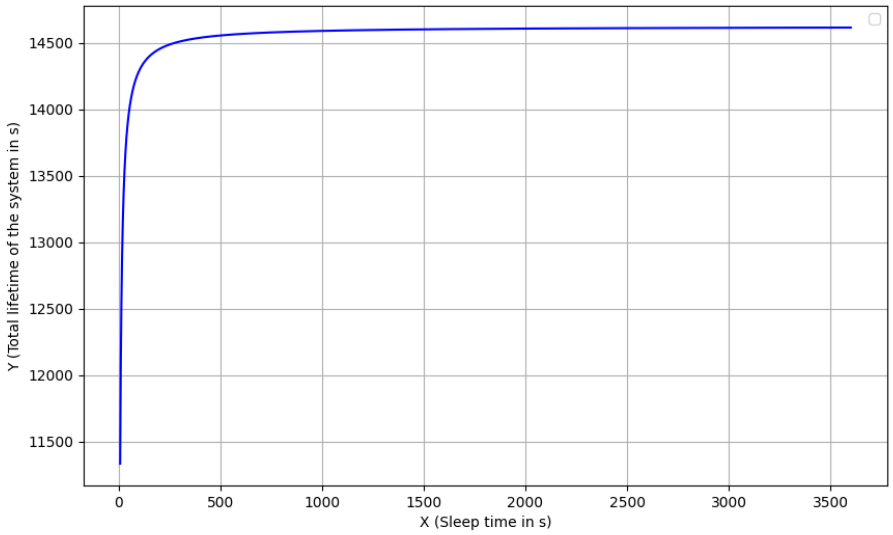

Figure 4: Total lifetime as a function of Deep Sleep Time.

The results showed that a good Deep Sleep period could be 500 seconds, improving significantly the overall system duration.

The total active system duration before and after optimization is shown below:

- Total lifetime duration before optimization: 10317.32 s
- Total lifetime duration after optimization: 14525.03 s
- Percentage improvement: **140.78**%

Key Result: Increasing the Deep Sleep interval to 500 seconds extends the system's operation by approximately 1.4 times, leading to better power efficiency.

5.2 Additional Optimization: Avoiding Redundant Transmission

Instead of sending data in every cycle, the ESP32 can compare the current parking status with the previously stored state:

- If the parking state has changed, the ESP32 transmits an update.
- If the parking state remains the same, the ESP32 skips the transmission and directly enters Deep Sleep mode.

This method eliminates the **WiFi activation cost** and the **transmission energy consumption**, significantly extending the battery life of the node.

5.2.1 Considerations for the Sink Node

Since the ESP32 node may not send updates in every cycle, the **sink node** must account for this behavior. To ensure proper functionality, the sink node should assume that if no update is received, the last known parking status remains valid.

5.2.2 Expected Impact

By eliminating redundant transmissions, this optimization is expected to:

- Reduce energy consumption per cycle, especially in long-duration parking scenarios.
- Extend the operational lifetime of the sensor node without affecting functionality.
- Decrease unnecessary network traffic, improving overall efficiency.

5.3 Conclusion

By increasing the Deep Sleep period and optimizing the transmission logic, the energy consumption per cycle is significantly reduced, leading to an extended system lifetime. However, these improvements introduce a trade-off between power savings and system responsiveness. The choice of parameters should be carefully balanced based on the application's real-time requirements and reliability constraints to ensure optimal performance.