POLITECNICO
MILANO 1863

Internet of things

IoT Challenge #1
Wokwi and Power Consumption
Challenge PDF

Authors:
Daniel Shala - 10710181
Jurij Diego Scandola - 10709931

Academic Year:
2024 - 2025

1 Code deployment and modellization

1.1 Overview

This ESP32-based project utilizes an **HC-SR04 ultrasonic sensor** to measure distance, communicates via **ESP-NOW** protocol, and optimizes power consumption using **deep sleep mode**. The project detects if an object is within a predefined minimum distance and sends a status message to a receiver.

1.2 Code Breakdown

1.2.1 Pin Definitions and Constants

```
// Connect HC-SR04 TRIG pin to ESP32 pin 5
#define TRIG_PIN 5
// Connect HC-SR04 ECHO pin to ESP32 pin 18
#define ECHO_PIN 18

// Conversion factor us/s
#define uS_TO_S_FACTOR 1000000

// Duty cycle period X (based on person code calculation)
#define X ((81 % 50) + 5)

// Minimum free distance threshold (in cm)
#define MIN_FREE_DISTANCE 50

// MAC address of the receiver (ESP-NOW peer)
uint8_t broadcastAddress[] = {0x8C, 0xAA, 0xB5, 0x84, 0xFB, 0x90};

long duration;
int distance;
esp_now_peer_info_t peerInfo;
```

- Defines **TRIG_PIN** and **ECHO_PIN** for the HC-SR04 sensor.
- Calculates **X**, the duty cycle
- **MIN_FREE_DISTANCE**, is the threshold for detecting an occupied space.

1.2.2 Measuring Distance

```
void measureDistance() {
    // Ensure TRIG pin is LOW before sending a pulse
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(5);
```

```
    // Send a 10-microsecond pulse to the TRIG pin to start measurement
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);

    // Read the pulse width from the ECHO pin
    duration = pulseIn(ECHO_PIN, HIGH, 23000);

    // Convert duration to distance in cm
    distance = duration / 58;
}
```

- Sends a **10-microsecond pulse** to trigger the ultrasonic sensor.
- Uses **pulseIn()** to measure the time taken for the echo to return with a timeout set to 23 ms, which is about the time the sensor needs to read 400cm;
- Converts **time duration to distance** in centimeters.

1.2.3 Notifying the Master Device

```
void notifyMaster() {
    int status = distance <= MIN_FREE_DISTANCE;
    String msg = status ? "OCCUPIED" : "FREE";

    // Send message via ESP-NOW
    esp_err_t res = esp_now_send(broadcastAddress, (uint8_t*)msg.c_str(), msg.length() + 1);
}
```

- Determines if the **distance is below the threshold** and sets the status.
- Sends an **"OCCUPIED" or "FREE"** message to the master device using **ESP-NOW**.

1.2.4 Setup and Deep Sleep Configuration

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

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

    // Measure distance
    measureDistance();

    // Initialize ESP-NOW communication
    WiFi.mode(WIFI_STA);
```

```
    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        return;
    }

    // Register peer
    memcpy(peerInfo.peer_addr, broadcastAddress, 6);
    peerInfo.channel = 0;
    peerInfo.encrypt = false;
    if (esp_now_add_peer(&peerInfo) != ESP_OK) {
        Serial.println("Failed to add peer");
        return;
    }

    // Send update to master device
    notifyMaster();

    // Disable Wi-Fi to save power
    WiFi.mode(WIFI_OFF);

    // Enable deep sleep for X seconds
    esp_sleep_enable_timer_wakeup(X * uS_TO_S_FACTOR);
    esp_deep_sleep_start();
}
```

- **Initializes Serial Communication** for debugging.
- **Configures pins** for the HC-SR04 ultrasonic sensor.
- **Measures the distance** using `measureDistance()`.
- **Sets up ESP-NOW**:
- Initializes **ESP-NOW communication**.
- **Disables Wi-Fi** after sending data to save power.
- **Configures deep sleep**:
- Uses a **timer-based wake-up** (`esp_sleep_enable_timer_wakeup()`).
- Calls `esp_deep_sleep_start()` to put the ESP32 into low-power mode.

2 Energy Consumption Estimation

The battery energy is computed using the following formula:

$$Y = (ABCD \mod 5000) + 15000 \quad \text{[Joule]} \tag{1}$$

where the leader personcode is defined as:

$$\text{Leader personcode} = 1069ABCD \tag{2}$$

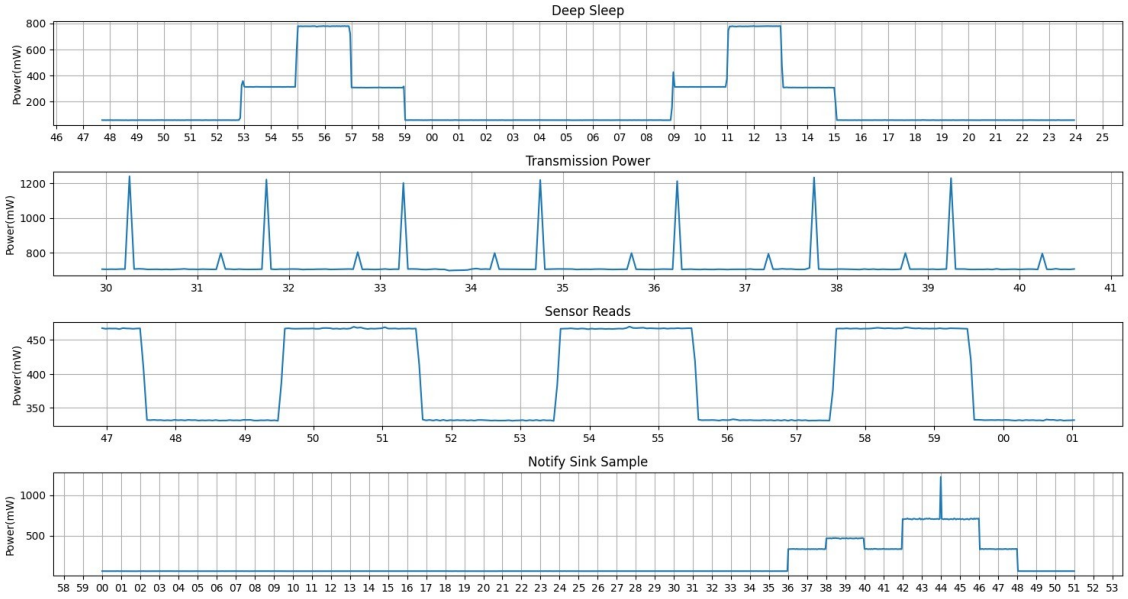
For our case, the team leader's personal code is **10710181**, so the total battery energy is equal to:

$$Y = 15,181 \text{ J} \tag{3}$$

And the duty cycle will be equal to:

$$X = 81 \mod 50 + 5 = 36 \text{ s} \tag{4}$$

By running two different Python scripts and utilizing the `matplotlib` library, we generated power-over-time graphs for each phase of the process and subsequently calculated the overall power of the sample test. Finally, the last graph represents the sample test, which integrates all phases into a single visualization.



Average Power

$$P_{ave} = (1 - f_{active})P_{sleep} + f_{active}P_{active} \tag{5}$$

$$P_{ave} = f_{sleep}P_{sleep} + f_{wakeup}P_{wakeup} + f_{work}P_{work} \tag{6}$$

2.1 Estimation of the average Power consumption

Deep Sleep

- **Deep sleep state**:

$$P_{\text{avg}} = \frac{\sum_{i=1}^{N} P_i \cdot \Delta t}{T} = 59.82 \text{ mW}$$

- **Idle/Boot/WiFi Off**:

$$P_{\text{avg}} = \frac{\sum_{i=1}^{N} P_i \cdot \Delta t}{T} = 310.64 \text{ mW}$$

- **WiFi On**:

$$P_{\text{avg}} = \frac{\sum_{i=1}^{N} P_i \cdot \Delta t}{T} = 775.20 \text{ mW}$$

- **Avg states**:

$$P_{\text{avg}} = f_{\text{sleep}} P_{\text{sleep}} + f_{\text{idle}} P_{\text{idle}} + f_{\text{active}} P_{\text{active}} = 196.38 \text{ mW}$$

Transmission State

- **transmission state**:

$$P_{\text{avg}} = \frac{\sum_{i=1}^{N} P_i \cdot \Delta t}{T} = 724.25 \text{ mW}$$

Sensor Read

- **sensor read**:

$$P_{\text{avg}} = \frac{\sum_{i=1}^{N} P_i \cdot \Delta t}{T} = 394.65 \text{ mW}$$

Notes

When computing the average power in a system that operates in different states, we often use the weighted sum of power values corresponding to different operating modes. The weight assigned to each mode depends on the fraction of time spent in that mode.
Given that power is measured at discrete time intervals, the total time spent in a specific state can be expressed as the sum of all small time intervals Δt during which the system was in that state:

$$T_{\text{state}} = \sum \Delta t_{\text{state}}$$

The fraction of time spent in a given state is then:

$$f_{\text{state}} = \frac{\sum \Delta t_{\text{state}}}{T_{\text{total}}}$$

Since the average power is computed as the sum of power values weighted by their corresponding time fractions, we obtain:

$$P_{\text{avg}} = f_{\text{sleep}} P_{\text{sleep}} + f_{\text{idle}} P_{\text{idle}} + f_{\text{active}} P_{\text{active}}$$

Substituting $f_{\text{state}} = \frac{\sum \Delta t_{\text{state}}}{T_{\text{total}}}$, we get:

$$P_{\text{avg}} = \frac{\sum \Delta t_{\text{sleep}}}{T_{\text{total}}} P_{\text{sleep}} + \frac{\sum \Delta t_{\text{idle}}}{T_{\text{total}}} P_{\text{idle}} + \frac{\sum \Delta t_{\text{active}}}{T_{\text{total}}} P_{\text{active}}$$

This formulation ensures that each state's power contribution is proportionally weighted based on its actual time duration.

2.2 Energy consumption and battery duration

The energy calculation is based on the fundamental formula:

$$E = P \times \Delta t$$

where:

- E is the energy in joules (J),
- P is the power in watts (W),
- Δt is the time interval between each measurement in seconds (s).

Operation	**Time (µs)**
IDLE	1039
READ	23258
IDLE	196
WIFI	188392
TX	180
WIFI	8558
IDLE	171

Table 1: Delta Time Data for Each Operation

Justification:

1. **Power to Energy Relationship:** The formula $E = P \times \Delta t$ is the standard relationship used to calculate energy from power. Power represents the rate at which energy is used or transferred, and multiplying it by the time interval over which it is applied gives the total energy transferred or consumed.

2. **Conversion of Units:** Since the power in the dataset is given in milliwatts (mW), it must be converted to watts (W) to comply with the standard unit for energy (Joules). The conversion factor is:

$$1\,\text{mW} = 1 \times 10^{-3}\,\text{W}$$

Therefore, to convert the power from milliwatts to watts, we multiply the power value by 10^{-3}.

3. **Constant Time Interval:** The time interval between each sample, Δt, is constant and given as 0.05 seconds. This fixed time interval allows us to compute the energy for each sample by multiplying the power (in watts) by this fixed time interval.

4. **Summing Energy Contributions:** Since energy is calculated per sample, the total energy is obtained by summing the energy contributions from each sample. Thus, the total energy is:

$$E_{\text{total}} = \sum_i P_i \times \Delta t$$

where P_i is the power at each sample point and Δt is the fixed time interval.

In our case, applying a simple script to the dataset, we calculated that, each transmission cycle, is equal to **1114 mJ.**

Lifetime

$$\text{Lifetime} = \frac{\text{EnergyStore}}{P_{ave} - P_{gen}} \tag{7}$$

So, giving the battery energy of $\mathbf{Y = 15181\ J}$, we obtain a total amount of

$$\text{n} = \frac{15181\ \text{J}}{1.114\ \text{J}} = 13627\ \text{cycles} \tag{8}$$

3 Results commentary and improvement

3.1 Code implementation and commentary

3.2 Power Management Optimizations

1. **Deep Sleep Usage** → Reduces power consumption significantly.
2. **Wi-Fi Off After Transmission** → Saves energy after sending data.
3. **RTC Memory for Persistent Data** → Retains `lastValue` during deep sleep.

3.3 Optimizations

3.3.1 Conditional Compilation for Storage

```
RTC_DATA_ATTR int lastValue = -1;
```

- Uses `RTC_DATA_ATTR int lastValue = -1;`, which stores the value in RTC (Real-Time Clock) memory to persist it across deep sleep cycles.

3.3.2 Set timeout for sensor read

```
duration = pulseIn(ECHO_PIN, HIGH, MIN_FREE_TIMEOUT);
```

- `MIN_FREE_TIMEOUT` is the time the sound pulse from the HC-SR04 ultrasonic sensor should take to travel to an object and back if the object is at least 60 cm away. Since `MIN_FREE_DISTANCE` is 50 cm, we add a 10 cm margin to account for sensor inaccuracies and variations.

3.3.3 Check If a Notification Should Be Sent

```
if ((distance <= MIN_FREE_DISTANCE) != lastSent)
```

1. It checks if `distance` is less than or equal to `MIN_FREE_DISTANCE`, indicating that an alert condition exists.
2. If the current status differs from the last sent status, the function returns `true`, meaning a notification should be sent.

3.3.4 Sending Data via ESP-NOW

```
// Send data with esp_now
WiFi.mode(WIFI_STA);
WiFi.setTxPower(WIFI_POWER_2dBm);

if (esp_now_init() != ESP_OK) {
return;
}
```

```
// Register peer
memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.channel = 0;
peerInfo.encrypt = false;

if (esp_now_add_peer(&peerInfo) != ESP_OK) {
return;
}

String msg = distance <= MIN_FREE_DISTANCE ? "OCCUPIED" : "FREE";

esp_err_t res = esp_now_send(broadcastAddress, (uint8_t *)msg.c_str(), msg.length() + 1);

WiFi.mode(WIFI_OFF);
```

1. **Enable WiFi in station mode** (`WIFI_STA`) to initialize ESP-NOW.
2. **Initialize ESP-NOW** (`esp_now_init()`):

- If initialization fails, the function exits.

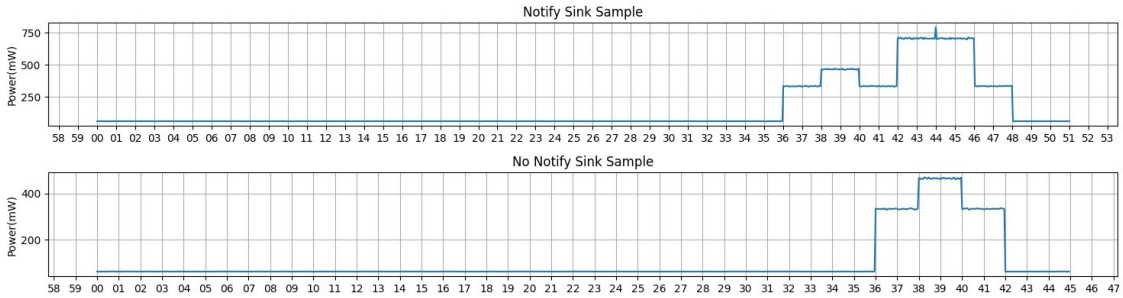
3. **Register the peer**:

- Copy the broadcast address to `peerInfo.peer_addr`.
- Set the communication channel to `0` and disable encryption.
- Add the peer using `esp_now_add_peer()`. If this fails, the function exits.

4. **Send the notification using** `esp_now_send()`.
5. **Disable WiFi** (`WIFI_OFF`) after sending the data to save power.

This structure optimizes power consumption and ensures efficient event-based notifications using ESP-NOW.

3.4 Energy consumption differences



In this case, the energy consumption is equal to **1100 mJ.**

The graph is just a visualization of the whole cycle. Idle and TX times would be too small to be visible in a true-to-scale representation.

Lifetime

$$\text{Lifetime} = \frac{\text{EnergyStore}}{P_{ave} - P_{gen}} \tag{9}$$

So, giving the battery energy of $\mathbf{Y = 15181\ J}$, we obtain a total amount of

$$\text{n} = \frac{15181\ \text{J}}{1.100\ \text{J}} = 13800\ \text{cycles} \tag{10}$$

We achieved an improvement of approximately 173 cycles. Furthermore, if the node does not detect any change in the last transmitted status, it skips the notification to the sink and enters deep sleep mode before even turning on Wi-Fi. This optimization results in additional energy savings, further extending the system's lifetime.

- In this case the energy consumption is **966,41 mJ.** and the number of cycles is

$$n = \frac{15181\ \text{J}}{0.96641\ \text{J}} = 15708\ \text{cycles}$$

Statistically, we can expect the actual value to fall somewhere between these two extremes, assuming that the probabilities of the events "The parking lot is always occupied" and "The parking lot is always empty" are equal. This assumption implies a uniform distribution between these two scenarios, meaning that the parking lot's availability fluctuates in a balanced manner over time.
As we can observe, the implemented updates lead to an improvement in battery life. This is because, although maintaining a memory of the previous state may have an associated cost, from an energy perspective, reading and writing to memory is significantly less demanding than activating the Wi-Fi module and transmitting data via the antenna. Furthermore, from a practical standpoint, memory storage has become so affordable that it is no longer considered a significant trade-off.