POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

IOT Challenge 1

Author: **Matteo Leonardo Luraghi**

Person Code: 10772886
Academic Year: 2024-25

Contents

1 | Wokwi Parking Node

1.1. ESP32 Sensor Code

It’s possible to run the code on the Wokwi Simulator.

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

// pins of the HC-SR04 sensor
#define PIN_TRIG 12
#define PIN_ECHO 14

#define uS_TO_S_FACTOR 1000000
// person code: 10772886
// TIME_TO_SLEEP = (86 % 50) + 5 = 41s
#define TIME_TO_SLEEP 41

// MAC address of the receiver
uint8_t broadcastAddress[] = {0x8C, 0xAA, 0xB5, 0x84, 0xFB, 0x90};

esp_now_peer_info_t peerInfo;

unsigned long txEnd;

// callback function on message sending
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  // the message has been sent, it's possible to end the duration
     measurement
  txEnd = micros();
  Serial.print("Send Status: ");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Ok" : "Error");
}

// callback function on message receiving
void OnDataRecv(const uint8_t *mac, const uint8_t *data, int len) {
  Serial.print("Message received: ");
  char receivedString[len];
  memcpy(receivedString, data, len);
  Serial.println(String(receivedString));
}

void setup() {

  // --------------------------------------------------- SERIAL SETUP
  unsigned long idleStart = micros();

  Serial.begin(115200);

  // -------------------------------------------------- HC-SR04 SETUP

  // HC-SR04 setup
  // https://docs.wokwi.com/parts/wokwi-hc-sr04
  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECHO, INPUT);

  unsigned long idleEnd = micros();

  // ----------------------------------------- MEASUREMENTS MANAGEMENT

  unsigned long measureStart = micros();

  // start a new measurement
  digitalWrite(PIN_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG, LOW);

  unsigned long measureEnd = micros();

  unsigned long idle2Start = micros();
  // read the result of the measurement
  int duration = pulseIn(PIN_ECHO, HIGH);
  // convert the result in centimeters
  int distance = duration / 58;
  Serial.print("Distance in CM: ");
  Serial.println(distance);

  // ----------------------------------------------------- WIFI SETUP

  Serial.println("Turning wifi on...");

```

```
unsigned long idle2End = micros();

unsigned long wifiStart = micros();

// enable wifi
WiFi.mode(WIFI_STA);
delay(50);
// initialize the communication module
esp_now_init();

// setup callbacks for message sending and receiving
esp_now_register_send_cb(OnDataSent);
esp_now_register_recv_cb(OnDataRecv);

// peer registration
memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.channel = 0;
peerInfo.encrypt = false;
// add peer
esp_now_add_peer(&peerInfo);

// ------------------------------------------------- MESSAGE SENDING

// send message to the sink node
String message = (distance <= 50) ? "OCCUPIED" : "FREE";
unsigned long txStart = micros();
esp_now_send(broadcastAddress, (uint8_t *)message.c_str(), message.
    length() + 1);
// delay to show the successful message transmission
delay(10);

// ------------------------------------------------ SLEEP MANAGEMENT

// turn wifi off
WiFi.mode(WIFI_OFF);
unsigned long wifiEnd = micros();
unsigned long idle3Start = micros();
delay(50);

// set the wakeup after 41 seconds
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
unsigned long idle3End = micros();

Serial.print("Idle: "); Serial.println(idleEnd - idleStart + idle2End
    - idle2Start + idle3End - idle3Start);
  Serial.print("Measurement: "); Serial.println(measureEnd -
    measureStart);
  Serial.print("Wifi on: "); Serial.println(wifiEnd - wifiStart);
  Serial.print("TX: "); Serial.println(txEnd - txStart);

  Serial.flush();
  // start the deep sleep
  esp_deep_sleep_start();
}

void loop() {}
```

1.2. Code Logic Explanation

1.2.1. Overview

The code manages all its functionalities within the setup() function because the device enters deep sleep mode after execution and reinitializes upon waking. It includes two callback functions for message transmission and reception, which assist in debugging by printing the status of sent messages and displaying received messages (since the sink node is simulated using the broadcast address).

1.2.2. Measurement Management

To obtain a measurement from the HC-SR04 ultrasonic sensor, the trigger pin is set to HIGH for 10 microseconds. The duration of the returning signal on the echo pin is then measured and converted into distance by dividing the duration by 58, following the specifications from Wokwi documentation.

1.2.3. Wifi Setup

Once the measurement is acquired, the Wi-Fi module is activated. A brief delay ensures system stability before initializing the ESP-NOW communication protocol and registering the receiver as a peer.

1.2.4. Message Sending

Based on the measured distance, the device sends a message indicating whether the area is FREE or OCCUPIED. If the distance is 50 cm or less, it is considered OCCUPIED; otherwise, it is FREE.

1.2.5. Sleep Management

After transmitting the message, the Wi-Fi module is turned off. The device then schedules a wake-up event after 41 seconds and enters deep sleep mode.

2 | Power and Energy Consumption

2.1. Power Consumption

```
import json
import pandas as pd

def compute_transmission_power():
    df = pd.read_csv("transmission_power.csv")
    transmission_powers = {
        "tx_19dBm": df["Data"][df["Data"] >= 1150].mean(),
        "tx_2dBm": df["Data"][(df["Data"] >= 750) & (df["Data"] < 850)].
            mean(),
    }
    return transmission_powers

def compute_sensor_read():
    df = pd.read_csv("sensor_read.csv")
    sensor_read_power = {
        "sensor_reading": df["Data"][df["Data"] > 460].mean(),
    }
    return sensor_read_power

def compute_deep_sleep():
    df = pd.read_csv("deep_sleep.csv")
    deep_sleep_powers = {
        "deep_sleep": df["Data"][df["Data"] < 100].mean(),
        "idle": df["Data"][(df["Data"] >= 300) & (df["Data"] <= 320)].
            mean(),
        "wifi_on": df["Data"][df["Data"] > 750].mean(),
    }
    return deep_sleep_powers
```

```
if __name__ == "__main__":
    results = {
        "transmission_power": compute_transmission_power(),
        "sensor_read_power": compute_sensor_read(),
        "deep_sleep_power": compute_deep_sleep(),
    }

    print(json.dumps(results, indent=4))
```

Using the previous Python script, I calculated the average power consumption from the three provided CSV files for the following states:

- transmission at 19dBm: 1221,76 mW, computed from values $\geq$ 1150 mW
- transmission at 2dBm: 797,29 mW, computed from values $\geq$ 750 mW and $<$ 850 mW
- sensor reading: 466,74 mW, computed from values $>$ 460 mW
- deep sleep: 59,66 mW, computed from values $<$ 100 mW
- idle: 310,88 mW, computed from values $\geq$ 300 mW and $\leq$ 320 mW
- wifi on: 776,62 mW, computed from values $>$ 750 mW

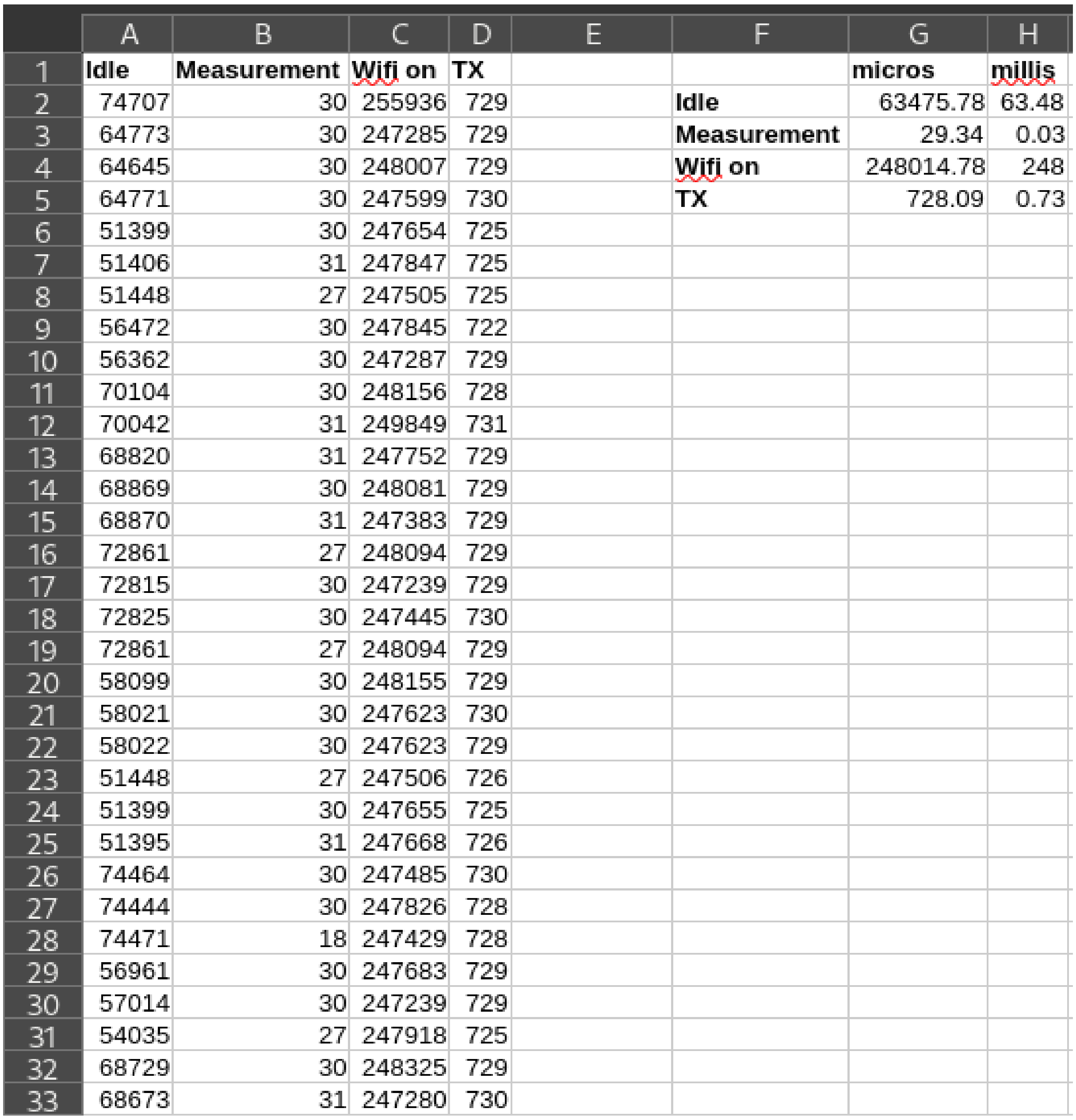
These threshold values were used to filter and categorize the data before computing the average for each state.

2.2. States Duration Measurements

Using the *micros*() function in Wokwi, I measured the average duration of each state:

- transmission at 19dBm: 728,1 μs
- sensor reading: 29,34 μs
- idle: 63,48 ms
- wifi on: 248 ms

Here's an example of the data I gathered:



	A	B	C	D	E	F	G	H
1	Idle	Measurement	Wifi on	TX			micros	millis
2	74707	30	255936	729		Idle	63475.78	63.48
3	64773	30	247285	729		Measurement	29.34	0.03
4	64645	30	248007	729		Wifi on	248014.78	248
5	64771	30	247599	730		TX	728.09	0.73
6	51399	30	247654	725				
7	51406	31	247847	725				
8	51448	27	247505	725				
9	56472	30	247845	722				
10	56362	30	247287	729				
11	70104	30	248156	728				
12	70042	31	249849	731				
13	68820	31	247752	729				
14	68869	30	248081	729				
15	68870	31	247383	729				
16	72861	27	248094	729				
17	72815	30	247239	729				
18	72825	30	247445	730				
19	72861	27	248094	729				
20	58099	30	248155	729				
21	58021	30	247623	730				
22	58022	30	247623	729				
23	51448	27	247506	726				
24	51399	30	247655	725				
25	51395	31	247668	726				
26	74464	30	247485	730				
27	74444	30	247826	728				
28	74471	18	247429	728				
29	56961	30	247683	729				
30	57014	30	247239	729				
31	54035	27	247918	725				
32	68729	30	248325	729				
33	68673	31	247280	730				

2.3. Energy Consumption

The energy consumption in each state is determined by multiplying the power consumption by the duration of that state:

- transmission at 19dBm: $E_{TX} = 1221,76\ mW * 728,1\ \mu s = 889,56\ \mu J$
- sensor reading: $E_{SR} = 466,74\ mW * 29,34\ \mu s = 13,69\ \mu J$
- deep sleep: $E_{DS} = 59,66\ mW * 41\ s = 2,45\ J$
  (the deep sleep state lasts 41 seconds as specified in the project instructions)
- idle: $E_I = 310,88\ mW * 63,48\ ms = 19,73\ mJ$
- wifi on: $E_W = 776,62\ mW * 248\ ms = 192,6\ mJ$

Summing up all contributions, the total energy consumed per transmission cycle is:

$E = E_{DS} + E_I + E_{SR} + E_W + E_{TX} = 2,66\ J$

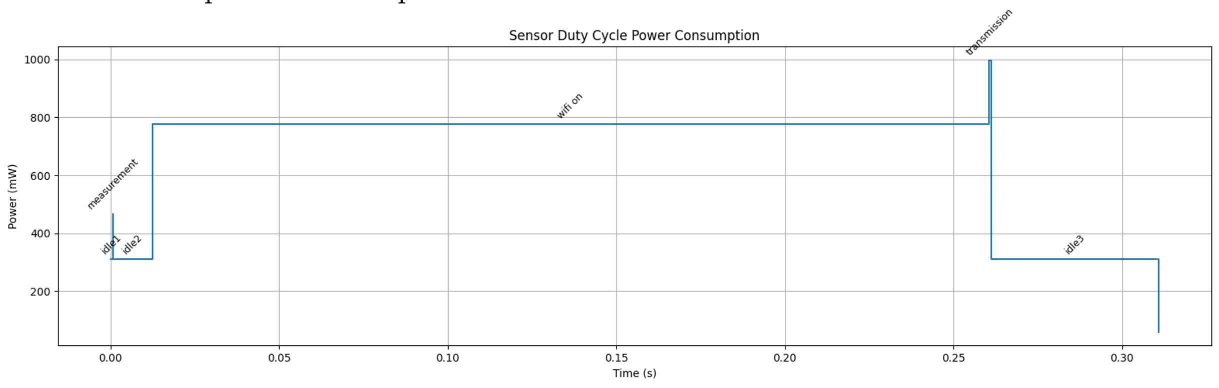
The sensor node is powered by a battery with an energy capacity of:
$(2886 \mod 5000) + 15000 = 17886\ J$

The number of transmission cycles the battery can sustain is: $\frac{17886}{2{,}66} = 6724$

Since the deep sleep state ($41\ s$) is significantly longer than the other time intervals, we approximate the total battery life as: $6724 * 41s = 275684\ s \approx 76$ hours.

2.4. Duty Cycle

The following is an estimation of the duty cycle of the sensor given all of the previous durations and power consumptions of each state:

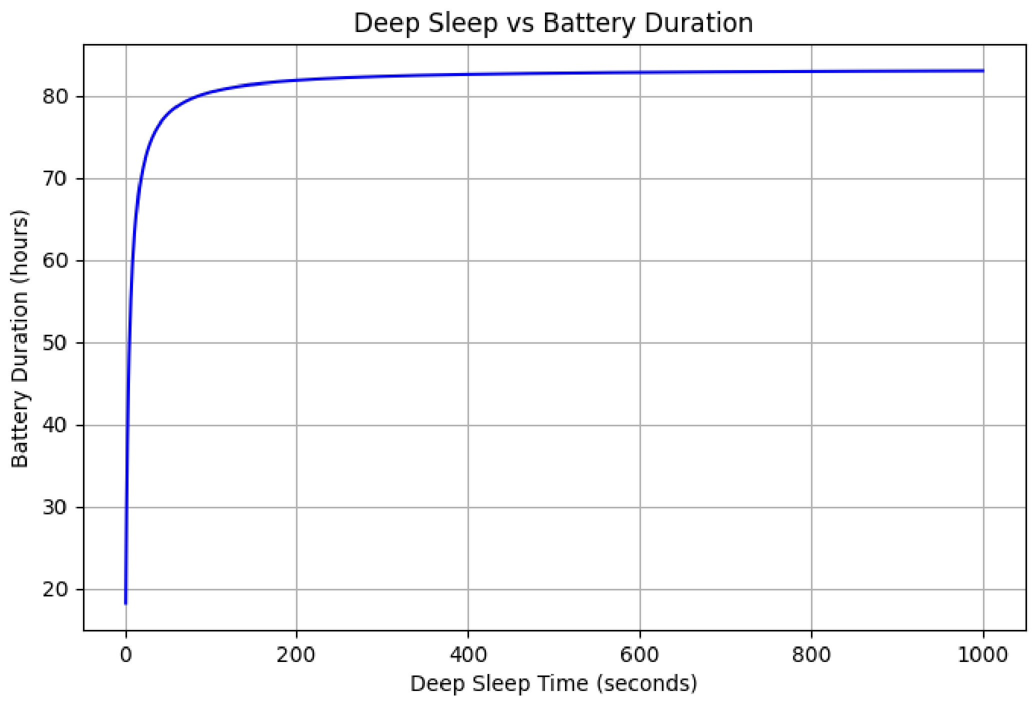


3 | Improvements

3.1. Deep Sleep Optimization

One effective way to extend the sensor node's battery life is by increasing the duration of its deep sleep state.

Using the previous formulas, we can plot battery lifetime against deep sleep duration. The results clearly show that beyond 90 seconds of deep sleep, any further increase provides negligible improvement in battery life.



Python script used to generate the plot:

```python
import numpy as np
import matplotlib.pyplot as plt

def formula(x):
    # compute the energy consumption of each cycle
    # 0.213 is the energy consumption of transmission, wifi on, sensor
       reading and idle
    energy_consumption = (0.05966 * x) + 0.213
    # compute the number of cycles before the battery is fully drained
    number_of_cycles = 17886 / energy_consumption
    # compute an approximation of the lifetime of the battery in seconds
    battery_lifetime_seconds = number_of_cycles * x
    # compute and return the lifetime of the battery in hours
    battery_lifetime_hours = battery_lifetime_seconds / 3600
    return battery_lifetime_hours

X = np.linspace(1, 1000, 1000)
Y = formula(X)

plt.figure(figsize=(8, 5))
plt.plot(X, Y, color='b')
plt.xlabel('Deep Sleep Time (seconds)')
plt.ylabel('Battery Duration (hours)')
plt.title('Deep Sleep vs Battery Duration')
plt.grid()
plt.show()
```

Increasing the deep sleep duration to 90 seconds provides a battery life improvement. This duration also maintains an acceptable balance between energy efficiency and real-time system responsiveness. Increasing deep sleep to two minutes or more offers only marginal battery gains while potentially compromising the system's real-time effectiveness.

Energy consumption in the case of 90 seconds of deep sleep:
$E_{DS} = 59,66\ mW * 90\ s = 5,37\ J$
Hence the total energy consumed per transmission cycle is:
$E = E_{DS} + E_I + E_{SR} + E_W + E_{TX} = 5,6\ J$

The number of transmission cycles the battery can sustain is: $\frac{17886}{5,6} = 3193$, using the same approximation as before, the total battery life is: $3193 * 90s \approx 79$ hours.

3.2. Efficient Status Reporting

Since increasing the deep sleep duration provides only a slight improvement in battery life, it may not be the most effective optimization. Instead, a better approach is to enhance the sensor node's energy efficiency reducing unnecessary Wi-Fi usage by utilizing RTC memory to store the last recorded status of the parking spot.

The updated duty cycle will be as follows:

1. the ultrasonic sensor will measure the distance to determine if the parking spot is OCCUPIED or FREE
2. the new status will be compared to the last recorded status stored in RTC memory
3. if the status remains unchanged, the node will skip Wi-Fi activation and return to deep sleep immediately
4. if the status has changed, the node will turn on the WiFi, initialize the ESP-NOW communication, transmit the updated status to the sink node and finally turn the WiFi off before entering deep sleep

The sink node will operate under the assumption that each parking spot is in the last reported state. It will only receive notifications when a change in status occurs, rather than receiving updates at fixed 41-second intervals.

I measured the average duration of each state, and the results closely match those from Chapter 2. The key difference is that if the parking spot's status remains unchanged, the device avoids Wi-Fi activation and transmission, reducing power consumption.

In the worst case, where the parking spot changes status every 41 seconds, the estimated battery life remains the same as in Chapter 2 (approximately 76 hours). However, in more realistic scenarios where the status remains unchanged for multiple cycles, the battery will last longer.

If, for example, we assume that the status changes with a probability of 50%, only half of the cycles will use the total energy needed which was $2,66J$, the other half will only consume $E_{SR} + E_{DS} + E_I = 2,47J$
To compute the number x of cycles that will drain the battery:
$2,66 * x * 0.5 + 2.47 * x * 0.5 = 17886$
$x = \frac{17886}{0.5*(2,47+2,66)} = 6973$ cycles, which means the battery will last $6973 * 41s \approx 79$ hours.

If the probability of changing status is less than 50%, then the lifetime of the battery will increase.

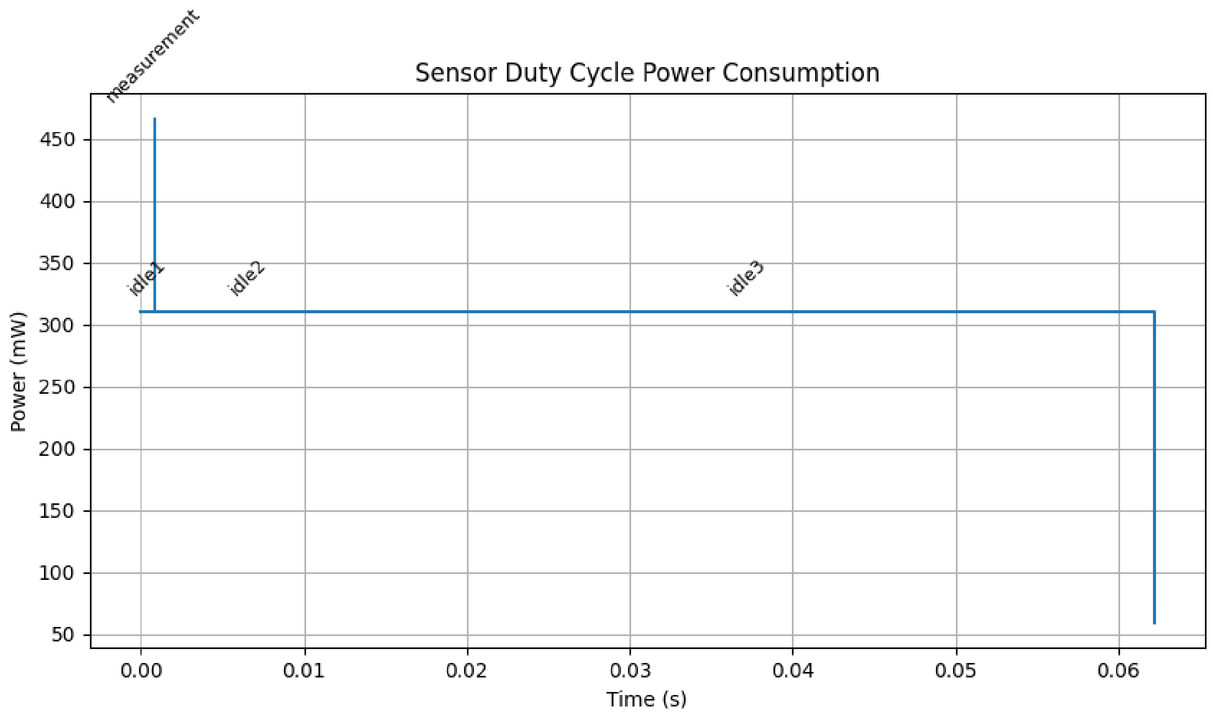
3.2.1. Simulation

Another Wokwi project is available to simulate the new behaviour.

I used RTC_NOINIT_ATTR instead of RTC_DATA_ATTR because the latter does not retain its stored value during simulation, in this way the simulation resembles more the behaviour that the sensor would have if it were actually running.

3.2.2. Duty Cycle

The following is an estimation of the duty cycle of the sensor given all of the previous durations and power consumptions of each state in the case that the status doesn't change and there is no wifi and transmission states:


Sensor Duty Cycle Power Consumption
measurement
idle1
idle2
idle3
Power (mW)
Time (s)