POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

IoT Challenge #1, Wokwi and Power Consumption

Internet of Things

Authors: **Kevin Ziroldi - 10764177**

Matteo Volpari - 10773593

Professors: Alessandro Redondi, Antonio Boiano
Academic Year: 2024-2025
Version: 1.0
Release date: 20-3-2025

Contents

Contents **i**

1 Code explanation **1**
1.1 Global variables . . . 1
1.1.1 Auxiliary variables . . . 1
1.1.2 Pin declaration . . . 1
1.1.3 Exercise constants . . . 2
1.2 Auxiliary functions . . . 2
1.3 Setup function . . . 3

2 Energy consumption estimation **5**
2.1 Power consumption estimation . . . 5
2.1.1 Power consumption plots . . . 5
2.1.2 Average power consumption values . . . 8
2.2 States duration estimation . . . 10
2.3 Energy consumption estimation . . . 12

3 Improvements **15**

List of Figures **17**

List of Tables **19**

1 | Code explanation

In this section, we will describe the main structure of the code written to meet the specification provided to us.

1.1. Global variables

1.1.1. Auxiliary variables

We have defined two global variables that are not strictly necessary for the correct operation of the system but are useful for future points of the challenge.

```
#define DEBUG 0
```

Variable which, when set to 1, is used in testing to print the messages arriving in broadcast to the board and the status of the messages sent. This variable also introduces a small delay of 10 ms to print the messages before going into deep sleep.

```
#define TIME_MEASUREMENT 1
```

This variable, when set to 1, is used to measure all the inter-times of the various phases of a system operation cycle. To function correctly, *DEBUG* must be set to 0, otherwise the data collected is distorted by the delay introduced by *DEBUG*.

1.1.2. Pin declaration

```
#define TRIG 4
#define ECHO 2
```

We have defined 2 variables which identify the functional pins used by HC-SR04 sensor and which we have connected to pins 4 and 2 of the ESP32 board.

1.1.3. Exercise constants

```
#define TIME_TO_SLEEP 48
```

Variable used to define the time for which the board will go into deep sleep. The value 48 is calculated using the formula: $TIME\_TO\_SLEEP = 93\%50+5 = 48$
where 93 is coming from the person code 10773593.

```
#define uS_TO_S_FACTOR 1000000
```

Constant equal to 10^6 used to convert from seconds to μs.

```
#define DISTANCE_LIMIT 50
```

Threshold distance of 50 cm set by the exercise to consider the parking space occupied or not.

1.2. Auxiliary functions

We have used various auxiliary functions to make the code more readable and functional, which we will describe in this section.

```
void setupESP_NOW()
```

This function is used to switch on the WiFi and do the setup. In the setup phase, we set the transmission power to 2 dBm and assign two callback functions which are used during debugging and triggered when messages are sent or received (*OnDataSent* and *OnDataRecv*).

```
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)
void OnDataRecv(const uint8_t * mac, const uint8_t *data, int len)
```

These are two callback functions which are used in debugging to print out whether a message has been sent correctly or whether there have been errors and all messages arriving on the board.

```
float getDistanceSensorMeasurement()
```

This function is used to perform a single measurement via the ultrasonic sensor. The

measurement is made by setting the trig pin to high, after which a delay of 10 μs is waited and the value of the trig pin is reset to low.
The delay is required by the HC-SR04 component specification, which needs the *TRIG* pin to remain high for at least 10 μs in order to make a measurement. After taking the data it converts it into cm and returns it.

1.3. Setup function

The setup function is the heart of the system and is executed by the board every time it wakes up. In our implementation, the setup contains all the actions performed by the board, since the loop is empty. We have chosen not to use the loop function because, after being sent into deep sleep, the board will have to start over from setup.
First of all, the setup function sets the data rate for serial data transmission to write to the serial. Then, the modes of the two pins *TRIG* and *ECHO* are set to output and input respectively. After this, a measurement is made via the ultrasonic sensor and, only then, the *setupESP_NOW()* function is called to activate WiFi.
When the WiFi is switched on and ready to be used, a message is sent to the MAC address of the sink, which in this case is set to the broadcast address 8C:AA:B5:84:FB:90.
Once the measurement is sent, the WiFi is switched off to reduce the energy consumption and the system is sent into deep sleep. On awakening from deep sleep, the board will restart from the beginning of the setup function and will then perform all the operations already seen in the same order.

2 | Energy consumption estimation

2.1. Power consumption estimation

In this section, there is a description of the power consumption of the ESP32 node in all states: deep sleep, idle, sensor reading, WiFi turned on and transmission.
In order to estimate power consumption based on the provided CSV files, we first plotted data using Python and Matplotlib library. Based on the plot, we computed the average power consumption in the analyzed working state.

2.1.1. Power consumption plots

In order to plot the data contained in the provided CSV files, we used the following algorithm.

```
import pandas as pd
import matplotlib.pyplot as plt
# read CSV file
df = pd.read_csv('deep_sleep.csv', parse_dates=['Timestamp'])
# plot data
plt.figure(figsize=(10, 6))
plt.plot(df['Timestamp'], df['Data'], linestyle='-', linewidth=2)
plt.xlabel('Time')
plt.ylabel('Power [mW]')
plt.title('Power consumption deep sleep')
plt.grid(True)
# save and shows
plt.savefig("power_consumption_deep_sleep.pdf", format="pdf",
                                    bbox_inches="tight")
plt.show()
```

Power consumption deep_sleep.csv

The following plot represents the power consumption when the ESP32 is in deep sleep state (lowest power level), goes into idle mode (medium power level) and finally turns the

WiFi on (highest power level).

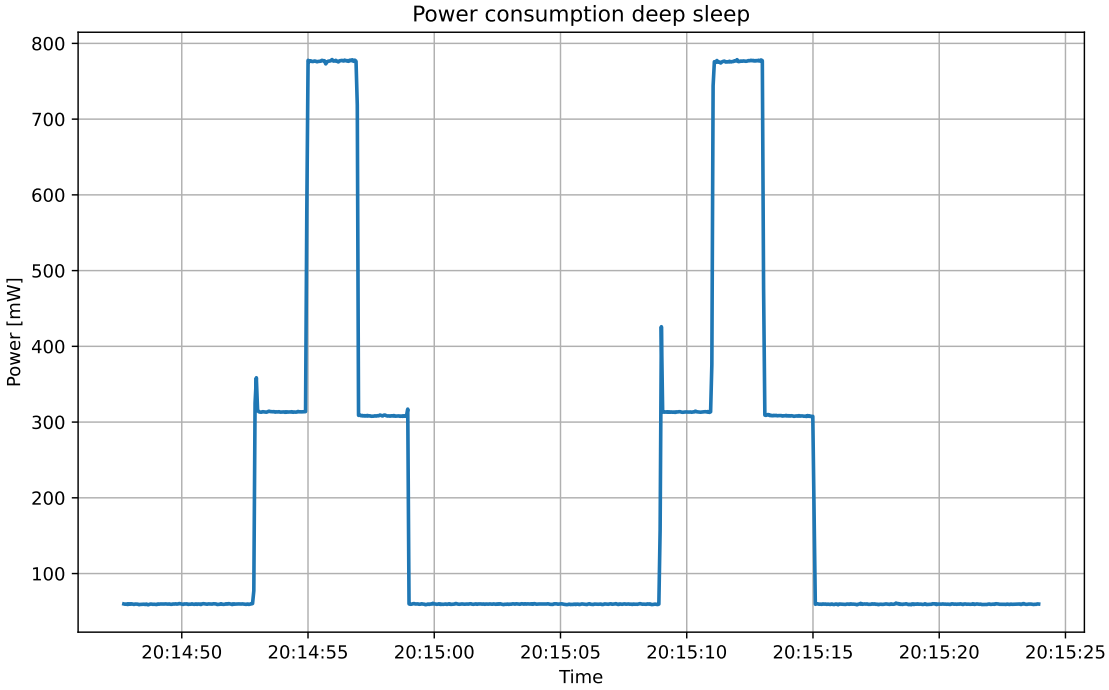

Figure 2.1: Power consumption deep sleep state

Power consumption sensor_read.csv

The following plot represents the power consumption when the ESP32 alternates the idle mode and sensor reading mode, in which it performs a measurement using the ultrasonic distance sensor HC-SR04.

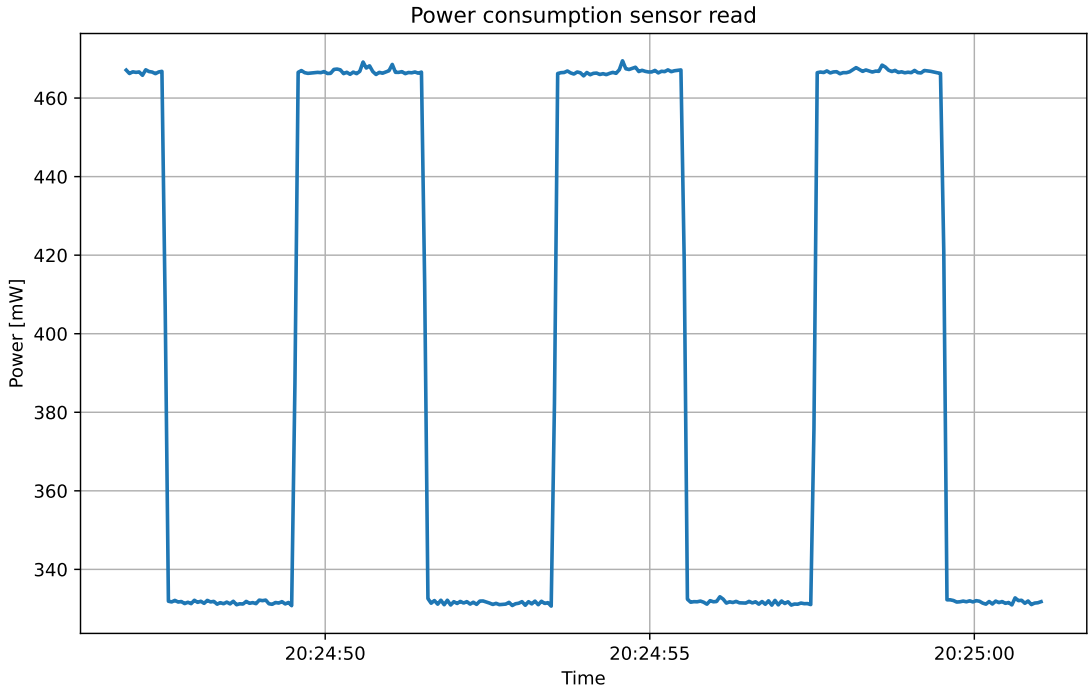

Figure 2.2: Power consumption deep sleep state

Power consumption transmission_power.csv

The following plot represents the power consumption when the ESP32 has the WiFi turned on and transmits data using ESP-NOW at 19.5 dBm and 2 dBm.

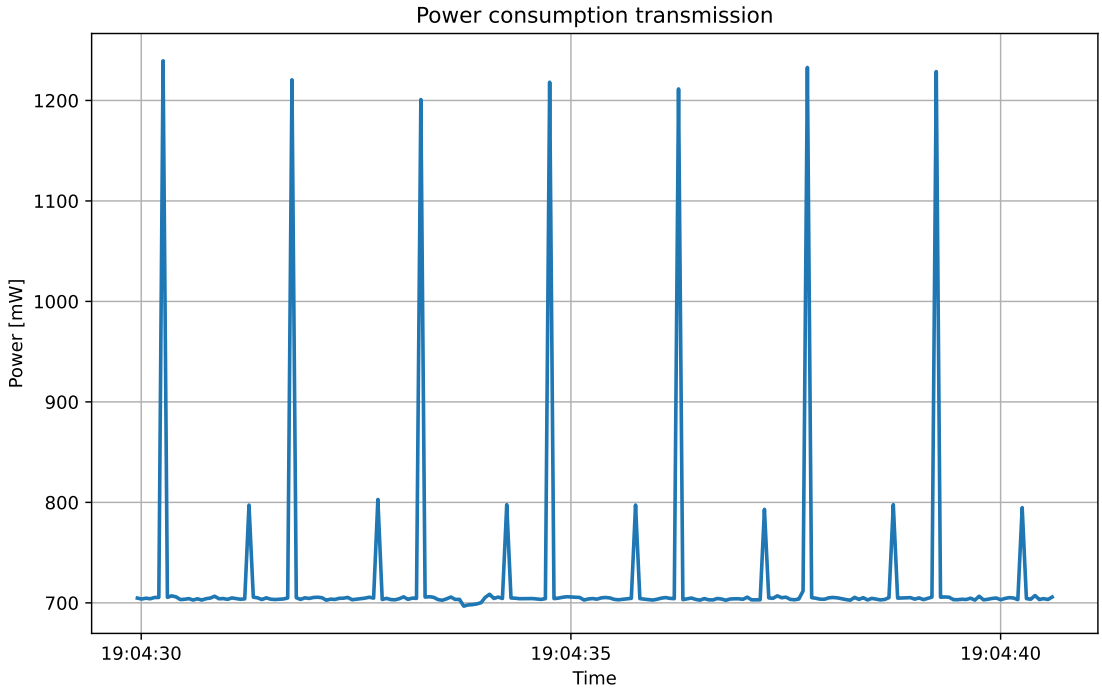

Figure 2.3: Power consumption deep sleep state

2.1.2. Average power consumption values

In order to find a numerical estimation of the power consumption in different states, we used some Python algorithms, using the library Pandas. Starting from the plot we selected the range of values regarding a particular state, discarding the others, and computed the average power consumption. For some states, the values of power can be found in multiple CSV files; in such cases, we merged different files in order to have a more accurate estimation.
In the following sections we also report two example of algorithms used to compute the average, both using a single dataset and multiple datasets.

Power consumption in deep sleep state

Data to estimate the deep sleep mode is contained in the dataset deep_sleep.csv. We filtered all values of power above 100mW, referring to other states as shown by the plot, and computed the average.

```
import pandas as pd
# read CSV file
dataset_deep_sleep = pd.read_csv("deep_sleep.csv", parse_dates=['
                                    Timestamp'])
# filter data with power < 100mW, referring to deep sleep mode
```

```
deep_sleep_data = dataset_deep_sleep[dataset_deep_sleep["Data"] < 100]
# compute average
deep_sleep_avg_power = deep_sleep_data["Data"].mean()
# print average
print("Average power consumption deep sleep: ", deep_sleep_avg_power)
```

We obtained an average consumption in deep sleep of 59.66 mW.

Power consumption in idle state

The power consumption in idle state can be extracted both by sensor_read.csv and transmission_power.csv. We used both datasets, merging all values and computing the average. We considered only values between 200 mW and 500 mW, that refer to idle mode.

```
import pandas as pd
# read CSV file
dataset_deep_sleep = pd.read_csv("deep_sleep.csv", parse_dates=['
                                    Timestamp'])
dataset_sensor_reading = pd.read_csv("sensor_read.csv", parse_dates=['
                                    Timestamp'])
# filter data with power between 200 mW and 500 mW, referring to idle
                                    mode
idle_data_deep_sleep = dataset_deep_sleep[(dataset_deep_sleep['Data'] >=
                                     200) & (dataset_deep_sleep['Data']
                                    <= 500)]
# filter data with power <= 400 mW, reffering to idle mode
idle_data_sensor_reading = dataset_sensor_reading[dataset_sensor_reading
                                    ['Data'] <= 400]
# merge values
idle_merged = pd.concat([idle_data_deep_sleep, idle_data_sensor_reading]
                                    , ignore_index=True)
# compute average
idle_avg_power = idle_merged['Data'].mean()
# print average
print("Average power consumption idle: ", idle_avg_power)
```

We obtained an average consumption in idle of 322.62 mW.

Power consumption in measurement state

Data representing power consumption in measurement state is contained in the dataset sensor_read.csv. We computed the average value considering values of power above 400 mW and obtained an average value of 465.18 mW. The algorithm is very similar to deep

sleep mode, thus we don't report it.

Power consumption in WiFi on state

We estimated the power consumption when the WiFi is turned on and the ESP32 is not transmitting based on deep_sleep.csv and transmission_power.csv, merging all value between 600 mW and 750 mW, as shown for the idle mode. The average power consumption when WiFi is turned on is 724.58 mW.

Power consumption when transmitting at 2 dBm

Finally, we estimated the average power consumption when ESP32 is transmitting at 2 dBm, based on transmission_power.csv. We compute the average of values between 750 mW and 900 mW, obtaining an average value of 797.29 mW.

Summary power consumption

The summary of power consumption of ESP32 in different states is reported below.

State	Power
P_{idle}	322.62 mW
$P_{measurement}$	465.18 mW
P_{WiFi}	724.58 mW
$P_{transmission}$	797.29 mW
P_{deep_sleep}	59.66 mW

Table 2.1: States power table

2.2. States duration estimation

We estimated the duration of different states by simulating the ESP32 on Wokwi and measuring them. We saved the timestamps in different points of execution, computed the duration of each state and printed it on the serial log.
We simulated the execution for about a hundred cycles and saved the serial log. We provide few examples of entries of the serial log, in which the durations are in microseconds.

```
entry 0x400805dc
Idle duration: 882
```

```
Measurement duration: 24116
Sending duration: 64
WiFi duration: 189158
ets Jul 29 2019 12:21:46

entry 0x400805dc
Idle duration: 846
Measurement duration: 24212
Sending duration: 65
WiFi duration: 188548
ets Jul 29 2019 12:21:46

entry 0x400805dc
Idle duration: 845
Measurement duration: 24211
Sending duration: 64
WiFi duration: 187548
ets Jul 29 2019 12:21:46
```

We computed the duration of different states using a Python algorithm to extract the values from the log, using regexes, filtering outliers and computing the average.
In the measurement of the sending duration, during which the ESP32 uses WiFi, we noticed the presence of some values completely different from the others, caused by the simulation on Wokwi; we filtered them out to get a more accurate estimation.
Finally, we observed that the duration of the measurement with the HC-SR04 ultrasonic distance sensor varies based on the resulting distance: the higher the distance, the longer the duration. In order to have a realistic duration, we gathered data with different values of measured distance.

```python
import re
import statistics
# open log file
with open("time_measurements_log.txt", "r") as file:
    log_text = file.read()
# extract values from log using a regex
idle_values = list(map(int, re.findall(r"Idle duration:\s*(\d+)",
                                       log_text)))
measurement_values = list(map(int, re.findall(r"Measurement duration:\s
                                       *(\d+)", log_text)))
sending_values = list(map(int, re.findall(r"Sending duration:\s*(\d+)",
```

```
                                        log_text)))
wifi_values = list(map(int, re.findall(r"WiFi duration:\s*(\d+)",
                                        log_text)))
# filter wrong values, given by the simulation
wifi_values = [value for value in wifi_values if value <= 200000]
# compute averages
avg_idle = statistics.mean(idle_values) if idle_values else 0
avg_measurement = statistics.mean(measurement_values) if
                                        measurement_values else 0
avg_sending = statistics.mean(sending_values) if sending_values else 0
avg_wifi = statistics.mean(wifi_values) if wifi_values else 0
#print values
print("Average Idle duration:", avg_idle)
print("Average Measurement duration:", avg_measurement)
print("Average Sending duration:", avg_sending)
print("Average WiFi duration:", avg_wifi)
```

The deep sleep duration is computed from the last two digits of the person code, 10773593, as:

$$T_{deep\_sleep} = (93\%50) + 5 = 48\,\text{s}$$

The duration of different states is reported in the following table.

State	Duration
T_{idle}	833.70 μs
$T_{measurement}$	18647.79 μs
T_{WiFi}	188640.86 μs
$T_{transmission}$	59.60 μs
T_{deep_sleep}	48×10^6 μs

Table 2.2: States duration table

2.3. Energy consumption estimation

Starting from the power consumption and states durations estimations, we can easily compute energy consumed by the ESP32 by multiplying the power and the time of each state.

At start, the ESP32 is in idle mode for T_{idle}. Then it enters measurement mode to measure the distance with the ultrasonic distance sensor HC-SR04 for $T_{measurement}$. When

the measurement is completed, the ESP32 turns the WiFi on for T_{WiFi} and, during this same period, it transmits data at 2 dBm for $T_{transmission}$. Finally, it enters deep sleep for T_{deep_sleep}.

The energy for different states is computed as follows.

$$\begin{aligned} E_{idle} &= P_{idle} \cdot T_{idle} = 0.268\,\text{mJ} \\ E_{measurement} &= P_{measurement} \cdot T_{measurement} = 8.675\,\text{mJ} \\ E_{WiFi} &= P_{WiFi} \cdot \Big(T_{WiFi} - T_{transmission}\Big) = 136.642\,\text{mJ} \\ E_{transmission} &= P_{transmission} \cdot T_{transmission} = 0.048\,\text{mJ} \\ E_{deep\_sleep} &= P_{deep\_sleep} \cdot T_{deep\_sleep} = 2863.680\,\text{mJ} \end{aligned}$$

The energy consumption for a cycle is computed as the sum.

$$E_{cycle} = E_{idle} + E_{measurement} + E_{WiFi} + E_{transmission} + E_{deep\_sleep} = 3009.313\,\text{mJ}$$

The available energy is computed from the last four digits of the person code, 10773593, as:

$$E_b = (3593\%5000) + 15000 = 18593\,\text{J} = 18593000\,\text{mJ}$$

The sensor node has a lifetime, measured in cycles, of:

$$L_{cycles} = E_b / E_{cycle} = 6178.487\,\text{cycles}$$

The total time for a cycle is:

$$T_{cycle} = T_{idle} + T_{measurement} + T_{WiFi} + T_{deep\_sleep} = 48.208\text{s}$$

The lifetime, measured in time, is of:

$$L_{time} = L_{cycles} \cdot T_{cycle} = 297852.501\text{s}$$

The lifetime is equivalent to 3 days 10 hours 44 minutes and 12 seconds.

3 | Energy consumption improvements

In order to improve the energy consumption of this IoT system, it is possible modify multiple parameters:

- transmission power for ESP-NOW communication;
- delay used to make a measurement with HC-SR04;
- time spent with WiFi turned on;
- time spent in deep sleep.

3.1. Improvements for our implementation

In our implementation, we use the lowest transmission power, of 2 dBm, and the lowest time, suggested by the documentation, of 10 μs to make measurement with HC-SR04 ultrasonic sensor. Finally, we perform the board setup and the measurement with WiFi turned off.
Thus, the only parameter which we can modify to improve the system's lifetime is the deep sleep time. We can tolerate a longer time to update the status of occupancy of the parking spot, but achieve a better lifetime of the node.
For instance, we can adopt a deep sleep time of two minutes and compute the new energy consumption and lifetime of the system.
The new energy consumption, with $T_{deep\_sleep} = 120s$, for the deep sleep phase is:

$$E_{deep\_sleep} = P_{deep\_sleep} \cdot T_{deep\_sleep} = 7159.2\,\text{mJ}$$

Since all other values of energy consumptions are unchanged, we can compute the energy consumption of a cycle as:

$$E_{cycle} = E_{idle} + E_{measurement} + E_{WiFi} + E_{transmission} + E_{deep\_sleep} = 7304,833\,\text{mJ}$$

The new sensor node's lifetime, measured in cycles, is:

$$L_{cycles} = E_b / E_{cycle} = 2545.301 \text{ cycles}$$

The total time for a cycle is:

$$T_{cycle} = T_{idle} + T_{measurement} + T_{WiFi} + T_{deep\_sleep} = 120.208\text{s}$$

The lifetime, measured in time, is of:

$$L_{time} = L_{cycles} \cdot T_{cycle} = 305965.563\text{s}$$

The new lifetime, with $T_{deep\_sleep} = 120s$, is equivalent to 3 days 12 hours 59 minutes and 25 seconds, which is more than 2 hours longer than the base case.

3.2. Improvements upper bound

In this section we prove that it is not possible to improve the lifetime of our system by a lot, by computing the lifetime if the deep sleep time tends toward infinity, i.e. the node always stays in deep sleep and isn't functional anymore.
The power consumption of the ESP32 in deep sleep mode is $P_{deep\_sleep} = 59.66\,\text{mW}$ and the battery life is $E_b = 18593\,\text{J}$, so the lifetime is:

$$L_{time} = E_b / P_{deep\_sleep} = 311649.346\text{s}$$

The lifetime of the system, when the deep sleep time tend towards infinity, is of 3 days 14 hours 34 minutes and 9 seconds, which is less than four hours bigger than the base case and less than two ours bigger than the improved version of the system.