Internet of Things – First Challenge

Francesco Pastore 10629332

Alongside this report file, you will find my Wokwi project, a Jupyter Notebook used for data analysis, the provided CSV files for use with the notebook, and an additional file containing the project's timing measured in Wokwi.

https://wokwi.com/projects/392275585925782529

1. Project Details

The Wokwi project is based on an ESP32 board connected to an HC-SR04 ultrasonic distance sensor, which is used to measure occupancy. The measurement is then checked and based on the value one of two possible messages is sent via WiFi and ESP NOW. After that, the device goes into deep sleep, waiting for the next cycle.

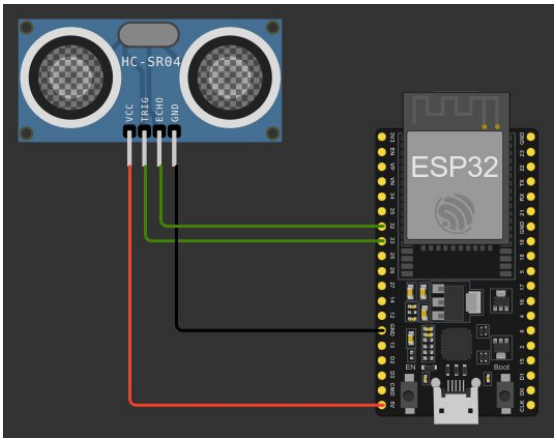
The general flow of the program can be summarized in the following steps:

1. Configuration of the pins for the HC-SR04. (idle)
2. Initiating a new distance measurement with the sensor. (sensor reading)
3. Checking the distance to determine the appropriate message to be sent. (idle)
4. Activating and configuring WiFi and ESP NOW for transmission. (WiFi On)
5. Sending the message. (TX)
6. Deactivating WiFi. (WiFi Off)
7. Setting a timer for wakeup. (idle)
8. Entering deep sleep mode. (deep sleep)

This process is repeated according to the required duty cycle. In this case it was calculated as 32 % 50 + 5 = 37s

I decided to use 2 dBm transmission instead of 19.5 dBm transmission because of the lower power required. This decision was made to prioritize energy efficiency and minimize power consumption during the project. However, if the sink node is too far away, problems may occur and more transmission power may be required.

The project worked well without delays, showing the correct measurements and no errors while sending the message. However, to see the received message the only way was to add a small delay before turning off the WiFi. This should be tested in a production environment to verify that messages are sent without errors.



2. Power and Energy Consumption

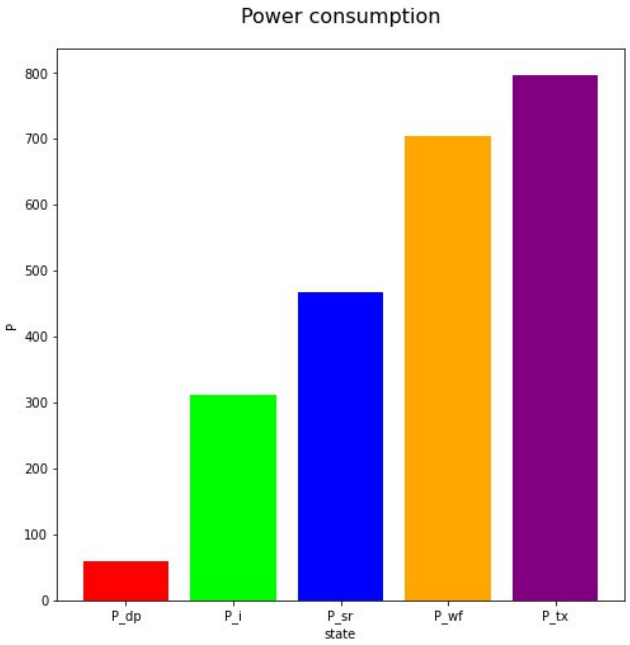
For this part, you can find the details of all the calculations inside the Jupyter Notebook.

2.1 Power Consumption

For power consumption analysis, I utilized the provided CSV files to estimate the power consumption of the ESP32 card during the different operational states.

Values were calculated by filtering the CSV files within the range of each state and computing the mean value to provide an accurate estimate of power consumption.

File	Name	Value	Description
deep_sleep.csv	P_dp	59.62 mW	Power in deep sleep state
	P_i	310.97 mW	Power in idle state
read_sensor.csv	P_sr	466.74 mW	Power for sensor reading
send_different_TX.csv	P_tx	797.29 mW	Power for transmission at 2 dBm
	P_wf	704.22 mW	Power when WiFi is on



2.2 Time Estimation

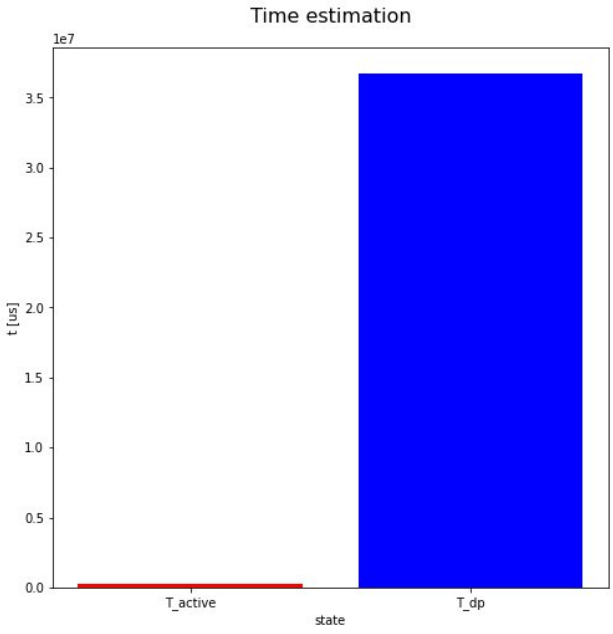
For time estimation, I used the micros function within the Wokwi project to record the start and end times of each phase. I took ten measurements and calculated the average time for each step.

During this process, I saw some cycles where the values deviated significantly from the norm, which I considered not useful for standard estimation conditions and thus excluded from consideration.

I also observed that different runs of the same code had the same identical time values, indicating that Wokwi's simulation parameters remain unchanged by default. This is consistent with the values used for estimation over multiple runs.

I report the final results for each step below:

Name	Value	Description
T_i	247 ms	Time in idle state
T_sr	916 ms	Time for sensor reading
T_tx	489 ms	Time for transmission at 2 dbm
T_wf	9 ms	Time when wifi is on
T_active	257 ms	Time when the device is active
T_dp	36742 ms	Time in deep sleep state
T_tot	37 s	Total time



2.3 Energy Consumption

For the calculation of the energy consumption I consider the power and time estimation calculated before.

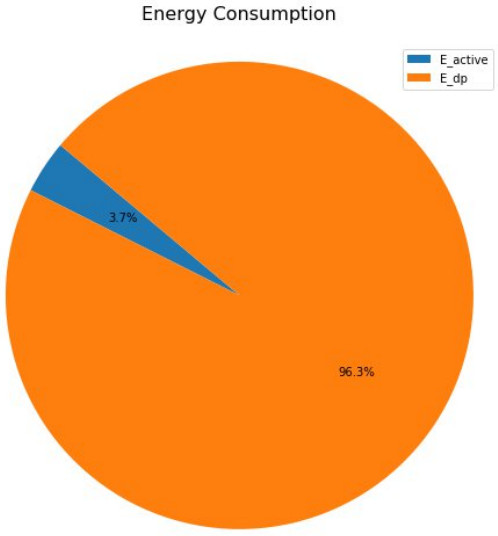
The most expensive state is deep sleep, simply because the system stays there for a long time. The other states seem to be cheaper, such as the transmission state, which lasts for a very short time. However, it still consumes a high amount of energy compared to the others.

Name	Value	Description
E_i	76.84 mJ	Energy spent in idle state
E_sr	0.427 mJ	Energy spent for sensor reading
E_tx	0.39 mJ	Energy spent for transmission at 2 dbm
E_wf	6 mJ	Energy spent when wifi is on
E_active	83.7 mJ	Energy spent when the device is active
E_dp	2190 mJ	Energy spent in deep sleep state
E_tot	2274.3 mJ	Total energy spent by one cycle

Based on the formula given, the battery capacity is 9332 +5 = 9337 J.

With a total energy consumption of 2274.3 mJ per cycle, we can expect approximately 4105 cycles from the battery.

Given that each cycle lasts 37 seconds, the total operating time would be 151885 seconds, which is equivalent to approximately 105 days.



3. Possible Improvements for Energy Consumption

3.1 Longer Duty Cycle

The most energy efficient state is deep sleep. Since we are monitoring human activity, considering also common parking behavior, 37 seconds may not be necessary.

Longer intervals, such as 60 or 90 seconds, may be sufficient to ensure that the system is able to react to any change in parking occupancy.

So, a longer duty cycle could be used, again depending on the needs and usage of the parking lot. That way we might get a few more days of battery life.

Below are some calculations I did with longer duty cycles, taking into account the previous results. In the Jupyter notebook you can find all the calculations that I have done.

Duty cycle	E_dp	E_tot	Days
60s	3561.87 mJ	3645.56 mJ	106.7 days
90s	5350.47 mJ	5434.16 mJ	107.4 days
120s	7139.07 mJ	7222.76 mJ	107.8 days
150s	8927.67 mJ	9011.36 mJ	107.9 days

3.2 Sending Only Changes in Status

A parking space might be vacant for many hours, such as overnight, and then be occupied for the entire workday. This means that the status of an individual parking space might change a few times a day.

Transmission is one of the most expensive parts of the active state. So if we can avoid this step, we could gain additional battery time.

Instead of sending the same message multiple times, we could send only a new message each time the status changes. For redundancy, we might send two or three messages before waiting for the next switch.

3.3 Longer sleep after a new occupancy

When a car is parked in a slot, it often stays there for a long period of time. We usually come back only after a few minutes at least, even if we have to do a quick activity.

To optimize power usage, we can implement a dynamic deep sleep duration that increases when a new parking event occurs or when the car remains parked for a specific duration.

By extending the deep sleep duration after the status changes from free to occupied, unnecessary sensor readings and transmissions can be avoided, conserving energy effectively.

3.4 Code Optimization

We could continue to keep the same flow, but avoid unnecessary function calls, such as printing the timestamps used for time estimation, which could help reduce the final power consumption.

The code could also be further optimized to avoid any waste of resources and time. For example, callbacks for sending and receiving messages could be avoided.

3.5 Transmission Power

For this project, I utilized the available 2 dBm power provided by the ESP32 card. While this power setting couldn't be further optimized, it's important to note that it might pose limitations when attempting to cover longer distances.

Additionally, increasing the transmission power could significantly diminish the device's lifespan. So, it should be done only if really necessary.

3.6 Different Transmission Technology

Instead of using WiFi and ESP NOW, we could look at other technologies such as Bluetooth Low Energy or Zigbee.

Using a different technology could result in a cheaper transmission step and longer battery life for the device.