Internet of Things

Challenge n. 1: Implementation report

Andrea Caravano, Alberto Cantele

Academic Year 2024–25

1 Implementation and design choices

The code is organized into small sub-functions to enhance clarity and generalize the behaviour with respect to the real board, also if not only emulated.
The overall structure is roughly described as follows. Deeper explanation is then expanded for the most relevant elements.

- The `loop` function is empty because of the employment of the Deep Sleep technique.

  Once the board reboots, in fact, the `setup` function is immediately called. Therefore, it is convenient to exploit it directly.

  The specification requires that a single measurement is sent by the node once every deep sleep interval (see following calculations).

- The `WiFi_initialization` function receives in input a peerStructure, used to register the peer that the sender node is connected to.

  In the emulation environment, it is represented by the virtual board itself, using the defined broadcast address.

  The Wi-Fi module of the board is turned on and prepared for usage via ESP-NOW (an ESP-32 specific implementation).

  The CALLBACKS functions are then registered for usage via the SEND or RECEIVE event activation, therefore triggered.

  Transmission power is finally set to 2 dBm, which is the minimum value for the parameter possible (see considerations in the energy consumption part).

- The `sendMessage` function simply uses ESP-NOW bindings to invoke the send action.

  The message is sent in textual form and converted to a byte array for network transfer, specifying the occupancy state of the parking slot.

- The `OnDataSent` callback function is called upon a send event via ESP-NOW and checks for transmission errors.

  In case transmission is successful, no serial prints are produced, as they are not useful in the emulation environment.

- The `OnDataRecv` callback function is called upon a reception event via ESP-NOW and plays the role of the sink node receiving the occupancy status of the parking slot.

  Result of the distance computation produced by the sender node is print in output via the Serial monitor.

- The `elaborateMessage` function produces the distance measurement via the HC-SR04 ULTRASONIC DISTANCE SENSOR.

As per the documentation examples, the measurement cycle requires to start setting initially the TRIGGER PIN to LOW, then expressing the intention to start a measurement writing an HIGH value for (at least) 10 μs.

The TRIGGER can finally be reset to LOW.

The distance measurement is proportional to the duration of ECHO PIN pulse, so the conversion factor is taken into account when doing the final evaluation and selection of the occupancy status.

- The `setup` function is executed each time the board is woken up from the Deep Sleep state or initialized for the first time.

  As usual, it prepares the Serial Console and the related Baud Rate.

  It specifies the PINs behaviour and links the other function invocations to the overall behaviour of the node, being detecting the parking occupancy status and informing the sink node of it.

  It finally computes the Wakeup timer for the Deep Sleep state and enters it, at the end of the function.

  It will be repeated at the board Wakeup.

2 Energy computations and plots

2.1 The datasets and the important figures of merit

In the Python notebook provided as a complement to the report, the biggest focus of the examination was put on the CSV files provided with the task's specifications, among a newly-crafted one, derived from the board's code instrumentation.
In fact, a particular constant value in the code enables instrumentation, upon toggling of the truth boolean value:

```
1 #define TIME_MEASUREMENT_INSTRUMENTATION true
```

The main aspects to consider in the consumption analysis are:

- Power consumption analysis: starting from the provided CSV files, averaging meaningful values among the ones concerning each algorithm's phase.
- Time estimation: starting from the detected measurements in the emulation environment, the goal is to determine the average duration of each algorithm's phase, excluding possible outliers, caused by the emulation overhead.
- Energy estimation: combination of the previous ones, resulting in the energy consumption by the board over time.

2.2 Power consumption analysis

In the provided dataset, the following figures of merit have been isolated.

- Deep Sleep state: The board enters a specific power-save status in which most components are turned off to minimize power consumption and therefore preserving battery lifetime.

  This is reflected in the following histogram, where theoretical expectations are met, resulting in the lowest possible power consumption state.

- Idle phase: The board is not providing useful computation, but its processing unit is still running and some components are still on, wasting some energy.

  The idle phase is composed of two parts:

  1. Between the beginning of the execution and the beginning of the Sensor Reading phase.
  2. Between the end the Sensor Reading phase and the Wi-Fi chip beginning its initialization.

- Wi-Fi Running phase: The read value is ready for transmission with ESP-NOW.

  The Wi-Fi chip and (emulated) peer structure is populated, the message is transmitted and the chip turned off.

  The chosen transmission power here is 2 dBm (see also the possible improvements section comment on the choice of a reasonable value).

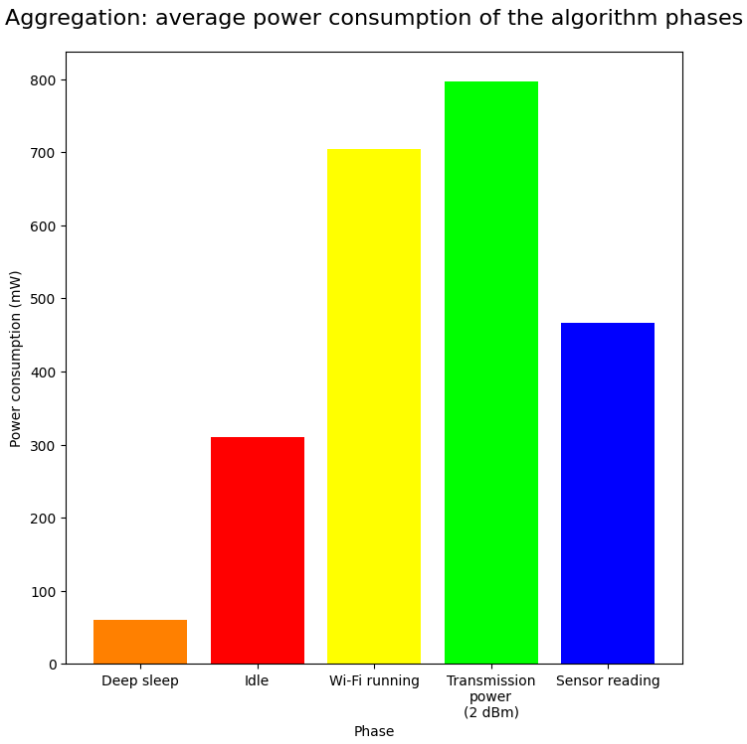
- Transmission phase: included in the Wi-Fi Running phase itself, being of course part of it, where the pre-computed message is transmitted using the defined transmission power.

- Sensor Reading phase: The HC-SR04 Ultrasonic Distance Sensor is activated and the pulse measurement is started (see earlier definition), resulting in the distance from the sensor, used to determine the final occupancy state of the parking slot.

Note that the transmission and Wi-Fi running phase are divided because of the different energy consumption: a peak in the consumed power is present in the Transmission phase.
As it will be clearly demonstrated via the following calculations, a huge importance is played by the time spent in each phase, rather than its peak value.
Average resulting values are summarized in the table below.

Power consumption analysis	
Deep Sleep state	59,62 mW
Idle phase	310,97 mW
Wi-Fi running phase	704,22 mW
Transmission phase (2 dBm)	797,29 mW
Sensor reading phase	466,74 mW

Overall, in the resulting histogram, theoretical expectations are completely met.



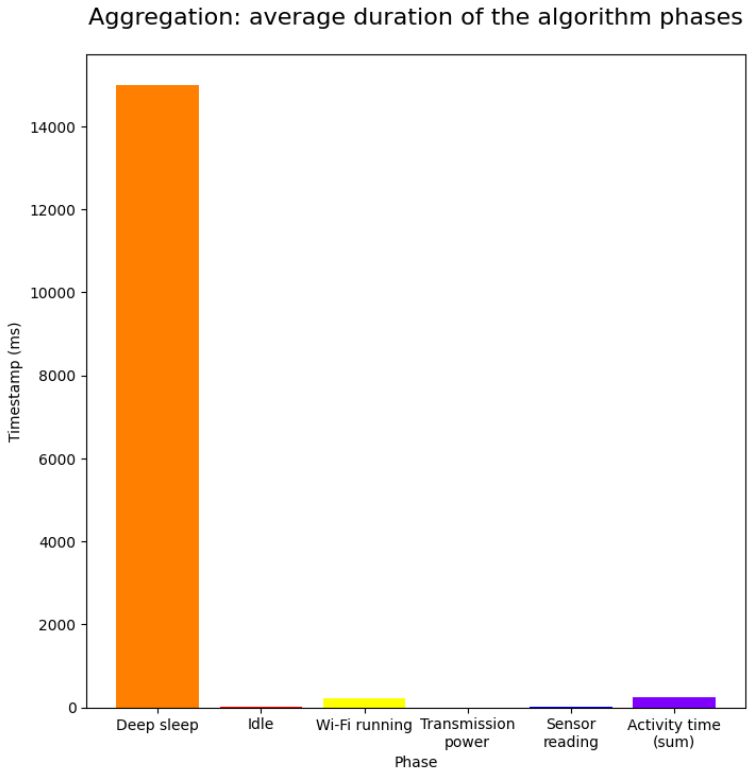
2.3 Time estimation

The most important part of the whole consumption detection job is the subdivision of the algorithm phases, isolating each sub-computation or state, also considering the sketched functional structure defined earlier.

Data is in fact aggregated according to differences computed between each phase: each one is made up of the mean of all the corresponding steps in all significative executions.
Finally, outliers caused by the emulation overhead are isolated.
Average resulting values are summarized in the table below.

Power consumption analysis	
Deep Sleep state (fixed)	15 seconds
Idle phase	2,44 ms
Wi-Fi running phase	219,64 ms
Transmission phase	0,30 ms
Sensor reading phase	24,10 ms
Total active time	246,18 ms

Overall, in the resulting histogram, theoretical expectations are completely met.



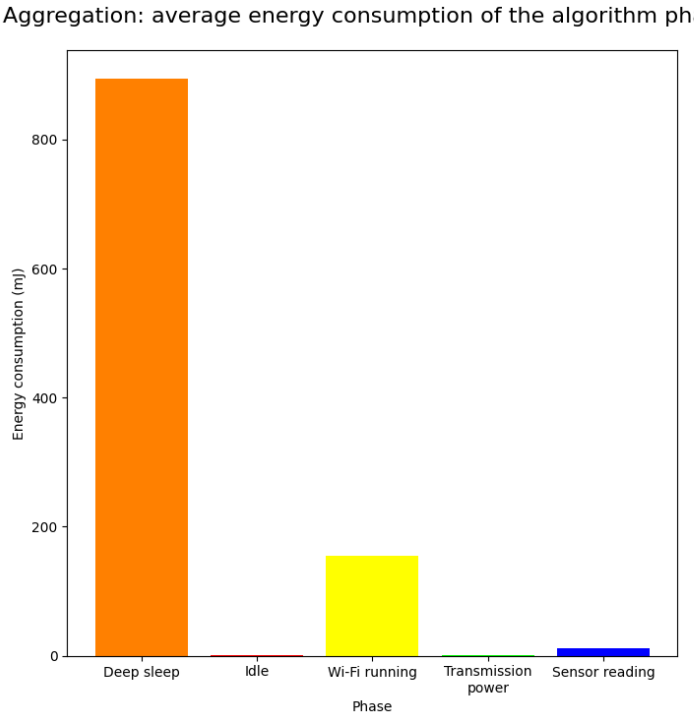
By looking at the histogram above, we may start to realize that, even if the Deep Sleep state is the least consuming state energy-wise, we still spend most of our measurement time in it, meaning the difference between the active and the sleeping state may actually turn out to be comparable, in an energy consumption setting.
Let's discuss it, by completing the estimation.

2.4 Energy consumption estimation

As a combination of the previous aggregations, the energy consumption is derived from the product of the time spent in each phase and its corresponding average power consumption.
Average resulting values are summarized in the table below.

Energy consumption analysis	
Deep Sleep state	894,30 mJ
Idle phase	0,76 mJ
Wi-Fi running phase	154,67 mJ
Transmission phase	0,24 mJ
Sensor reading phase	11,25 mJ
Total activity	166,92 mJ

The matching histogram is the following, confirming our previous suspects.



The Deep Sleep state, in fact, is the most prominent one energy-wise.
This should not come as a surprise, especially considering that it is the most time-consuming state, contributing with a huge difference with respect to overall time durations, even if less power-hungry, overall.
Let's now compute the resulting lifetime of the board.

Lifetime computation	
Energy budget	$8260\%5000 + 15000$ Joule $= 18260$ Joule
Resulting lifetime (cycles)	17206 cycles
Possible activity time	258090 seconds $\simeq 2 days$

Possible optimizations would, for sure, begin in the direction of reducing the usage of the Wi-Fi chip and are considered in the last part of the document.

3 Improvements proposals and possible revisions

3.1 The most sensible approach: A versioning system

As expected from the theory classes, the most energy-consuming element is confirmed to reside in the Wi-Fi usage (therefore, in the need of transmitting a sample per measurement).
If, instead, we exploit a non-volatile storage area to store past readings and compare them with a newly performed one, we would be able to send a new message only when a real difference was detected.
Natural evolutions would take into consideration the implementation field of the board: a few examples are shown in the following.

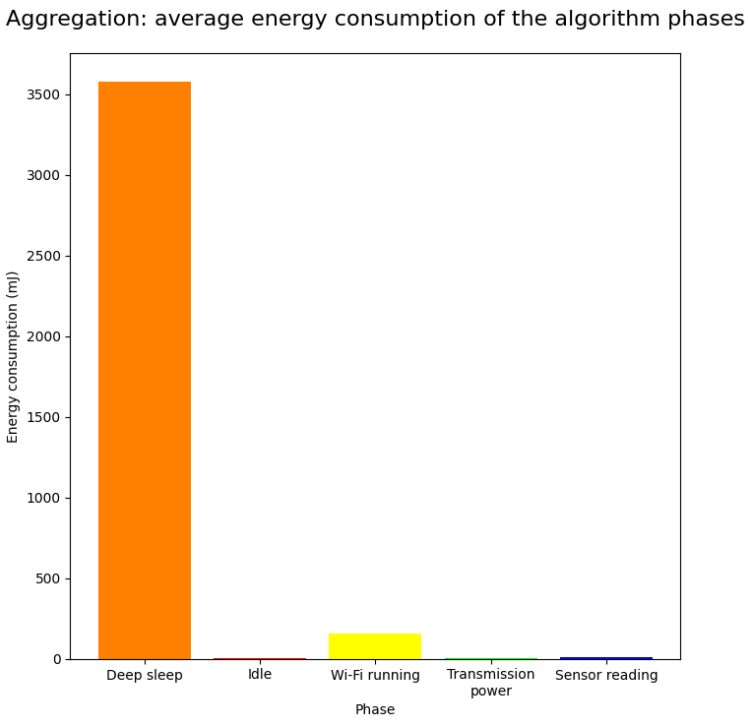
3.1.1 A higher duty cycle

The actual duty cycle, as shown, can be roughly approximated as $\simeq 15 seconds$.
As we are dealing with cars movements in a parking slot, changes in the occupancy state are definitely expected to be much less frequent than the duty cycle used in the current implementation.
Moreover, in the versioning system case, the duty cycle could be dynamically adapted to a recent change, behaving similarly to a sliding window update.

Lastly, an additional consideration about the usage of the occupancy states should be taken into consideration.
In a first approximation, a meaningful fixed value could be from 60 to 180 seconds, that are taken as a reference in the following.
Revising the attached Python notebook, we obtain the following lifetime figures.

60-seconds duty cycle	
Resulting lifetime (cycles)	4876 cycles
Possible activity time	292560 seconds $\simeq 3days$

Less possible cycles are completely expected, as they will last more. In fact, a higher activity time is a good sign of a meaningful optimization being achieved.
Overall, in the resulting histogram, theoretical expectations are completely met.



180-seconds duty cycle	
Resulting lifetime (cycles)	1675 cycles
Possible activity time	301500 seconds $\simeq 3days$

Here, we start to realize that a reasonable threshold has been reached by approaching to a 60 seconds duty cycle.
Therefore, we should stick to a value near to it (or adapt the measurement cycle correspondingly).

3.1.2 Parking activity times

As in most parking slots, regular activity times are generally observed, meaning a parking slot is expected to open at a certain morning time and close in the evening.
In the remaining time, no updates to the status of the slot are expected.
A specifically crafted night-cycle could in fact be implemented, expanding the duty cycle to a decidedly higher value ($\sim 30minutes, forexample$).
Also, remember that the node should keep its internal clock updated (for example, with the NTP protocol via the classical internet) and implement a mechanism to account for extraordinary opening and closure times.

3.1.3 Non-volatile storage

As shown during the laboratory lectures, a non-volatile variable can be retained among reboots. This mechanism can be exploited, for example, to store the latest distance measurement and the duty cycle currently in use.

```
RTC_DATA_ATTR float latest; // ex: 75.0 cm
RTC_DATA_ATTR int next_sleep_timer; // ex: 180 seconds
```

3.1.4 Keep-alive message

The protocol could finally include a new message serving as a heart-beat, for sensors that have not updated their status for at least a pre-defined amount of time, to distinguish low-usage periods of the slot from a battery drain, that would require maintenance.
An example is shown below. Note that, of course, an additional factor related to the activity time of the board should be considered.
In a first approximation attempt, it is accounted with a bonus cycle.

```
#define KEEPALIVE_FREQUENCY 30 * 60 // 30 minutes, expressed in seconds
...
RTC_DATA_ATTR int next_sleep_timer = 180; // seconds
RTC_DATA_ATTR int total_sleep_time; // seconds
...
String elaborateMessage() {
        ...
        if (/* no change in the occupancy status ... */ && (total_sleep_time +
        ↪  next_sleep_timer) > KEEPALIVE_FREQUENCY) {
            return "KEEPALIVE"; // The new message of the small application-layer
            ↪  textual protocol
        }
        ...
}
...
void setup() {
        ...
        // Before acting again, if needed, on the sleep_timer, it is recorded into the
        ↪  total_sleep_time
        total_sleep_time += next_sleep_timer;
        ...
}
```

3.2 Transmission power regulation

In the current implementation, the transmission power chosen is at 2 dBm.
For ESP-32 boards, this is the minimum value, naturally sufficient in the emulation environment, as the board itself acts as both a transmitter and receiver.
However, in the real implementation, an additional factor should be considered because of the distance of the two endpoints (sender and sink node).
As analyzed during the Energy computation part and in the laboratory classes, the maximum difference in energy consumption is $\sim 400mW$.

3.3 A different protocol stack

As just recently mentioned in the theory classes, standard network protocols may have a significant data overhead and, over time, this results as an added energy consumption component, which can't be ignored.

Using a different low-level transport and/or network protocol is expected to yield better results.
We may want to check at the system integration details first, as a best effort approach is going to provide a lower level of Quality of Service (think at the QoS 0 level case in MQTT, for example, even if the standard involved is of course different).

3.4 A final note: the role of the sink node as a star-centre

In the current setting, the sender node is the one expected to elaborate the direct relationship between the distance measured and the corresponding occupancy state.
Naturally, being the computation needed quite trivial, shifting the computation effort to the (supposedly, more powerful) sink node would not make a difference and would still perfectly respect the balance in the star network architecture, but if deeper computations are expected to be performed, a natural evolution could definitely benefit from the work shift.
Therefore, the sender node would ideally function only as a transmitter of the read value and not evolve the computation on it.