Elias Borge-Ilseth, 11075526
Eskil Myklatun Østbø, 11074713 20.03.2025

Internet of Things Challenge 1 Report

Wokwi link: https://wokwi.com/projects/425892411326510081

Task 1 – Parking occupancy node specification

The program is designed to run on an ESP32 microcontroller and implements a parking occupancy detection system using an HC-SR04 ultrasonic sensor and ESP-NOW wireless communication. The system periodically wakes up from deep sleep, measures the distance to detect a parked vehicle, transmits the occupancy status to a sink node, and then returns to deep sleep to conserve energy.

System overview of the program follows this sequence:

1. **Libraries**: import necessary libraries such as Arduino for microcontroller functions and ESP NOW for wireless communication.
2. **Definition of variables**: Defining the HC-SR04 pins as variables, creating a peer interface for the receiver (sink node) and defining the MAC address of the sink node, and other useful variables.
3. **Measuring distance**: help function for performing sensor reading with the HC-SR04. When "TRIG_PIN" is set to high for 10 microseconds, it sends a pulse towards the car. Then the echo duration is measured using "pulseIn(ECHO_PIN, HIGH)" function. The echo time of the pulse is converted into distance using the formula:

$$distance = \frac{duration * 0.034}{2}$$

   where distance between the sensor is calculated using duration of the pulse, the speed of sound through air, divided by two because the pulse travels a round-trip.
4. **Setup (main execution)**:
   a. **Initialization**: The ESP32 wakes up from deep sleep and enters booting phase.
   b. **Wi-Fi and ESP-NOW**: The ESP32 enables Wi-Fi in station mode and initializes ESP-NOW for communication. Initializes HC-SR04 sensor pins. Registers the sink node as a peer for ESP-NOW communication.
   c. **Sensor reading**: The HC-SR04 sensor measures the distance to determine parking occupancy, using the "measureDistance" function explained in step 3.
   d. **Transmission**: The ESP32 sends a message containing the occupancy status from the sensor reading via ESP-NOW.
   e. **Deep sleep**: The ESP32 sets a timer for 31 seconds, flushes serial output to avoid losing logs and enters deep sleep to conserve power.
5. **Empty loop()**: Since ESP32 goes into deep sleep after every transmission and resets on wake-up, loop() is never used.

Elias Borge-Ilseth, 11075526
Eskil Myklatun Østbø, 11074713 20.03.2025

Task 2 – Energy consumption estimation

Part 1 – Power Consumption

To estimate the average power consumption for each state of the sensor node, we developed a custom Python script. The provided CSV files contained timestamped power consumption data, which we analysed and classified into predefined states based on their value ranges. These ranges were determined through visual inspection of the plots associated with the data.

Our script, which is well-documented and included as *power.py*, processes the data by:

1. Grouping power consumption values into their corresponding states.
2. Calculating the average power consumption for each state.
3. Outputting the results in a clear tabular format for easy interpretation.

Our results can be seen in the table below:

State	Average Power Consumption (mW)
Deep-Sleep	59,66
Idle	320,91
Sensor Read	466,74
Wi-Fi on	724,58
Transmission at 2 dBm	797,29
Transmission at 19.5 dBm	1221,76

We will be able to use these values in the next section to determine how much energy the node spends during a duty cycle.

Note that there are multiple values that fall outside our defined ranges in the dataset. These are transient values and occur when moving from one state to another. We have chosen to not include these values as they do not directly correspond to a specific state. In addition, from the data we can see that for some states have multiple different power consumptions between cases. For example, we can see that in 'Transmission power' the 'Wi-Fi on' state has a value of around 700, but in 'Deep Sleep with Interrupt' the values are almost 800. In these cases, we are grouping all the data as the same state and calculating a global average.

Part 2 – Energy consumption

When estimating the energy consumption of the ESP32 we must measure the time the program spends in each of the states. To calculate this, we utilised the 'micros()' function and printed out the time differences between each section/state of the program. We would then let the program run for 50 cycles, where in one half the Ultrasonic distance sensor is set to 400cm and to 40cm in the other, to simulate that the space is occupied 50 percent of the time. We would then get a good estimate for how long an average cycle would take.

In the process, however, we noticed that some of the measurements had very large variances. Especially 'Wi-Fi on' where we had values ranging between 0.18 seconds and 50 seconds. We believe that these large discrepancies are a result of a bug in the simulation and decided to exclude outliers to not disturb our results. Most of the time the simulation spent around 180 milliseconds during the 'Wi-Fi on' phase and we removed results that was deviating from this. We then calculated the average times of each state.

The state times we calculated as part of one duty cycle were:

State	**Time Spent in State (μs)**
Deep-Sleep	31 000 000
Idle	80 500
Sensor Read	13 466
Wi-Fi on	185 339
Transmission	95

Our data can be found in the *timings.csv* file

Energy consumption [J] in each state of the duty cycle can be calculated as:

$$E_s = P_s \times t_s \text{ [J]}$$

where P is power consumption [W] in state and *t* is time [s] in state. Then, the energy consumption of 1 transmission cycle is calculated by summing the energy consumption of all states:

$$E_{cycle} = \sum E_s = 2.016 \text{ [J]}$$

The energy of the battery in a sensor is given as:

$$E_b = 5526 \ (mod\ 5000) + 15000 = 15\,526 \text{ [J]}$$

Thus, the number of complete duty cycles a node can perform is:

$$lifetime_{cycles} = \left\lfloor \frac{E_b}{E_{cycle}} \right\rfloor = \left\lfloor \frac{15\,526}{2.016} \right\rfloor = 7701 \text{ [cycles]}$$

From this we can get that the total lifetime of the node in seconds is:

$$lifetime = lifetime_{cycles} * t_{cycle} = 7701 \times 31.3 \approx 241\,000 \text{ [s]}$$

where t_{cycle} is the duration of 1 duty cycle. This gives us a total lifetime of about 2 days and 19 hours.

Task 3 – Improvements

Some possible improvements to reduce the energy consumption without modifying the main task of the parking sensor node are:

- Option 1: Reduce the transmission power when sending data from the sensor to the sink. By default, ESN-NOW transmits at full power of 19.5dBm, which is not necessarily needed if it can be reached while transmitting with a lower transmission power.
    - How to perform: Use “esp_wifi_set_max_tx_power(LOWER_TX_POWER)” to reduce transmission power. Set “LOWER_TX_POWER = 8” to reduce the transmission power to the lowest possible value 2 dBm.
- Option 2: Use RTC memory to store previous state such that the ESP32 does not use power to transmit a new message to the sink every time it wakes up, as it is unnecessary and wastes energy if no state change has occurred.
    - How to perform: To save data on the RTC memory, add “RTC_DATA_ATTR” in front of a variable definition, e.g. “RTC_DATA_ATTR int previousStatus = 0”, 0 indicating spot is empty, 1 indicating it is occupied. When waking up, compare the “previousStatus” to “currentStatus” to decide if a new transmission should be made or not.
- Option 3: Optimize wake-up frequency from a cyclic timer wake up to external motion-based interrupt wake up such that it only wakes up if there has been a change, most likely a car parking or leaving. This can be done by connecting an external sensor to the GPIO port of the ESP32. The external sensor is then able to send a hardware interrupt to the ESP32 to wake it up.
    - How to perform: esp_sleep_enable_ext0_wakeup(WAKEUP_GPIO, 1) will enable the ESP32 to wake up from deep sleep when input WAKEUP_GPIO goes to high.

After performing these optimization changes, we would expect to observe a reduced energy consumption in the results. The changes would reduce the power consumption in the transmission phase, reduce the energy consumption by cutting redundant transmission and lastly reduce energy consumption by avoiding waking up from deep sleep unless a motion sensor is triggered.