AY 2023/2024

Politecnico di Milano

Internet of Things Challenge - 1

Rishabh Tiwari - rishabh.tiwari@mail.polimi.it - 10987397
Marcos V. Firmino P. - marcosvinicius.firmino@mail.polimi.it - 10914211
Alexander Stephan - alexander.stephan@mail.polimi.it - 10932707

Professor
Redondi Alessandro Enrico Cesare

Version 1.1
March 19, 2024

Contents

1 Introduction **1**

2 Energy Calculations **1**

2.1 Per Mode Power Consumption 1

2.2 Energy Consumption for a Transmission Cycle 1

2.2.1 Assumptions 1

2.2.2 Time Measurements 2

2.2.3 Calculations 2

2.3 Time Until Battery Change 2

3 Description of Implementation **3**

4 Possible Improvements for Energy Consumption **4**

1 Introduction

In our project, we worked on an innovative Wokwi project to tackle a real-world challenge - monitoring parking spaces efficiently and sustainably. Our goal was to develop a simple yet effective parking occupancy sensor using the HC-SR04 ultrasonic distance sensor and ESP-NOW for communication. This sensor would determine if a parking spot is occupied and then communicate this information to a central hub. A key feature of our design was its energy efficiency; we programmed the sensor to enter a deep sleep mode, waking up periodically to check the parking space's status before going back to sleep. We also calculated the power and energy consumption for a full cycle of activity to estimate how long the sensor could operate on a single battery.
Additionally, we explored potential improvements to reduce energy consumption further without compromising the system's core functionality - ensuring the sensor's ability to notify the central hub about the parking spot's occupancy status. Throughout this project, we've navigated through coding challenges, energy consumption estimations, and thoughtful analysis on how to improve the system's efficiency, all while keeping the primary task in focus. This report outlines the logic behind our code, our findings on power and energy use, and our proposals for making the sensor node even more energy-efficient.

2 Energy Calculations

We used the given CSV files to make estimations of the energy consumed in each case. Then, we take the average out of the samples, importing the CSVs to Excel. The idle power is derived from the graph shown in the tutorial regarding energy consumption.

2.1 Per Mode Power Consumption

- Deep Sleep: $59.65\,mW$
- Idle: $331.57\,mW$
- Transmission State: $797.71\,mW$
- Sensor Readings: $466.75\,mW$

2.2 Energy Consumption for a Transmission Cycle

2.2.1 Assumptions

We can assume that the board goes directly to the *loop*, since the setup part is done only once, it is really fast and statistically insignificant for a system that should run for hours along.

Our leader has the following person code

10987397

, thus the calculated duty cycle is:

$$(97\%50)\,s + 5\,s = 52\,s \tag{1}$$

Our assumptions:

- The *boot* energy spike was not considered.
- No idle time was considered since we measured it to be insignificant during the loop. We only idle during the sensor measurement, but this is also factored-in the provided CSV.
- The time in deep sleep is equal to the duty cycle period 1, so 52 s.

2.2.2 Time Measurements

To retrieve the amount of seconds the system spent on each state, we deployed the `micros()` function, executing *deltas* in order to get the desired time frame. Given a distance of $45\,cm$—and this is important because the measuring time varies according to the distance - our results are:

- Time reading the ultrasound sensor: $3043\,\mu s$
- Time transmitting via Wi-Fi: $181\,\mu s$

2.2.3 Calculations

With all the information above, we can calculate the energy needed per cycle:

$$\text{Total Energy} = \text{Reading Time} + \text{Transmission Time} + \text{Sleeping Time} \tag{2}$$

Substituting the corresponding values

$$\begin{aligned}\text{Energy}_{\text{1Cycle}} &= 3043\,\mu s \times 466.75\,mW \\ +181\,\mu s \times 797.71\,mW &+ 52\,s \times 59.65\,mW\end{aligned} \tag{3}$$

Solving

$$\text{Energy}_{\text{1Cycle}} = 3.103\,J \tag{4}$$

2.3 Time Until Battery Change

The battery energy was calculated by the given formula, in our case:

$$Y = 7397 + 5 \tag{5}$$

$$\text{Battery Capacity} = Y\,J \mathrel{\widehat{=}} 7.402\,kJ \tag{6}$$

So, given the amount of energy spent in one cycle (see Equation 4), the number of cycles before a battery change will be:

$$N_{\text{cycles}} = \frac{\text{Battery Capacity}}{\text{Energy}_{\text{1Cycle}}} = \frac{7.402\,kJ}{3.103\,J} \approx 2386 \tag{7}$$

We assume that N_{cycles} is an integer. Note that we have to round up here, as in cycle 2385 there is still some energy left.

Given our duty cycle period, the total time will be:

$$N_{\text{cycles}} \times \text{Duty Cycle Period} = 2386 \times 52\,s = 124080\,s = 34.467\,h \approx 1.436\,d \tag{8}$$

So, according to our calculations, the system can stand alone for almost 1 day and a half (Equation 8).

3 Description of Implementation

The code logic for the sensor node goes as follows:

1. **Functions**: We wrote a handful of functions in order to organize the code and make it more readable.
   (a) `on_data_sent()`: performs a confirmation print whenever data is sent
   (b) `setup_ultrasonic_sensor()`: setup sensor's pins
   (c) `setup_wifi()`: setup Wi-Fi with
   $$2\,dB$$
   bandwidth
   (d) `register_peer()`: register the broadcast address
   (e) `register_hooks()`: registers callback for when data is sent
   (f) `print_results()`: print the timing information and the communication status
2. **Setup**:
   (a) Set the sleep time to $52\,s$ with `esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP*uS_TO_S_FACTOR);`. Note that we convert seconds to microseconds here according to the libraries' specifications.
   (b) Define and set the `TRIG` pin as output to trigger the measurement on the ultrasonic sensor.
   (c) Define and set the `ECHO` pin as input to perform the actual measurements.
   (d) Set up the Wi-Fi by setting the Wi-Fi mode to station mode and adding a send hook to validate the successful sending of the status.
   (e) Add a peer that is initialized with the protocol-specific broadcast address, so the library knows where to send the packets.
3. **Loop**:
   (a) Read the distance by setting the `TRIG` pin high for 10 ms and reading the value via the ECHO pin. Divide the retrieved value by 58, following the Wokwi documentation of this component.
   (b) Set the status according to the measured distance (depending on whether it is $\leq 50\,cm$).
   (c) Send status to the broadcast address via `esp_now_send()`.
   (d) Measures all the necessary timing.
   (e) Call deep sleep via `esp_deep_sleep_start()`. After the timer has ended, the loop repeat itself.

4 Possible Improvements for Energy Consumption

The requirement of the system is to perform correct measurements and deliver timely results while using as little energy as possible. This leads us to the following considerations:

1. The most obvious change is to disable Wi-Fi when not sending data. As we are only sending data to the network, but not receive data, listening for incoming packets and powering the Wi-Fi card is a waste of energy. So, after we read the data, we shortly enable Wi-Fi and then directly disable it again. This can be done with the default Wi-Fi library and is well described in online literature, e. g. `https://esp8266hints.wordpress.com/2017/06/29/save-power-by-reliably-switching-the-esp-wifi-on-and-off/` (applies also to ESP32).

2. With the current implementation, a value if sent out directly after its measurement. This leads to as many transmissions as there were measurements. However, for some applications, it might be acceptable that there is a delay in distance readings. In this case, we can batch measurements at the sensor node, e. g., send out 10 values at once. This could be achieved by appending the values to a comma-separated string. In practice, you probably could only batch 2–3 values because otherwise, the parking spot would become too inaccurate. You could also make a trade-off by lowering the duty cycle and increasing the batch size. This can lead to a more detailed occupancy detection, while still lowering the overall energy consumption.

3. E. g., overnight the parking spot might not be frequently used. In this case, the sensor will always read the same value. Same for cars that park for a long duration. Therefore, to save energy, we could only send out the values on a change, always comparing to the previous one. This can be used in combination with the first point.

4. Although we are in deep sleep after sending out the values, we still use a normal delay when reading the sensor. As already mentioned in the implementation section, there must be $10\,ms$ delay after triggering the sensor. A light sleep or even a deep sleep is possible here, further reducing the energy consumption of our system.

5. Another improvement would be to synchronize the sleep cycles. If the sensor node sends out its data, and the sink node is not available, the data needs to be re-transmitted. This consumes additional energy. As the exercise statement mentions that the energy consumption of the sink node is not restricted, we are unsure, if this improvement applies in our particular case since a sleep might not be needed at all. (Although the sink node has no energy constraints, it still might be desirable to reduce its energy consumption.)