Internet of Things
Challenge 1
Victor Medina - Juan Sebastián Mendez Hernandez - Mostafa Hashemiyan

POLITECNICO
MILANO 1863

Wokwi and Power Consumption

1. Parking occupancy node

For this challenge a parking occupancy node was emulated in Wokwi[1] with a ESP32 and a ultrasonic distance sensor HC-SR04 to measure the distance in a parking spot, as it is shown in *figure 1*. In that way, when the measurement is less than 50 cm there is a presence of a car, otherwise the parking spot is considered as free. This message is then sent to the ESP32 through WiFi protocol using esp_now.

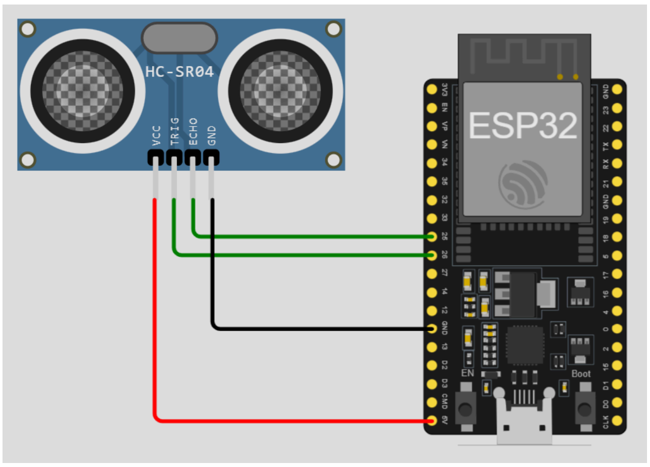

Figure 1. *Node schematic.*

It can be observed in *figure 1* the sensor HC-SR04 which has an input, TRIG, that is set high for at least 10 microseconds to indicate the start of a new measurement. The output, ECHO, is a pulse whose length is proportional to the measured distance, calculated in centimeters with the following function:

$$Distance = PulseMicros / 58 \ [cm]$$

Furthermore, for the development of the node, some functions were defined to achieve the measurement and the transmission of the message. First, *initializeNetwork()* enables the WiFi, initializes the Esp_now protocol and allows the Peer registration, which is used to reference the communication.

```
void initializeNetwork() {
  WiFi.mode(WIFI_STA); //Enable wifi
  esp_now_init(); //Initialize esp_now
  esp_now_register_send_cb(OnDataSent); //Send callback
  esp_now_register_recv_cb(OnDataRecv); //Receive callback

  memcpy(peerInfo.peer_addr, broadcastAddress, 6); //Peer Registration
  peerInfo.channel = 0;   // Channel number
  peerInfo.encrypt = false; // Encryption mode or not
  esp_now_add_peer(&peerInfo);
   WiFi.setTxPower(WIFI_POWER_19_5dBm);  //Set  transmission  power  to
19_5dBm WiFi.setTxPower(WIFI_POWER_2dBm);
}
```

[1] https://wokwi.com/projects/392700409621320705

Moreover, *distanceMeasure()* was implemented to set pin modes, start the measurement and read the result. This function returns a boolean that takes true as a value if the measured distance is greater than 50 cm, meaning the parking spot is free.

```
bool distanceMeasure() {
  int distance; //in cm

  // Start a new measurement:
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(Measurement_Time);
  digitalWrite(TRIG_PIN, LOW);

  // Read the result:
  int duration = pulseIn(ECHO_PIN, HIGH);
  distance = duration / 58; // in cm
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

   return distance >= DistanceTH; // True if Distance >= 50cm, which
means Free
}
```

For the transmission, the function `esp_now_send()` was used taking the broadcast addresses previously defined and sending the string message “Free” or “Occupied”. Likewise, the function `printingTimes()` was implemented to stamp the times corresponding to the sensor measurement and the transmission, as it is shown below.

```
void printingTimes(){
  Serial.print("Measure Time: ");
  Serial.print(MeasureEndingTime-MeasureStartTime);
  Serial.println(" us");
  Serial.print("Transmission Time: ");
  Serial.print(TransmissionEndingTime-TransmissionStartTime);
  Serial.println(" us");
  Serial.println("Awake time: " + String(awakeTime)+" ms");
}
```

For the calculation of the duty cycle, we use the personal code ending in 35, having a duty cycle specifically of 40 seconds to go to sleep mode. The value of *awakeTime* is used to calculate the time of sleep mode before going to sleep mode use. Additionally, In this part of the code, by using the duty cycle multiplied by the us_to_s_factor value, the amount of time the sensor is going to sleep is obtained. After this time the deep sleep starts .

```
 // Start going to deep sleep
 Serial.println("Going to sleep now");
 esp_sleep_enable_timer_wakeup(x * uS_TO_S_FACTOR);
 Serial.println("ESP32 sleep every " + String(x));
delay(2000);
 Serial.flush();
 delay(1000);

 printingTimes();

 esp_deep_sleep_start(); //Deep sleep time
```

2. Energy Consumption Estimation

This section aims to determine the energy consumption of various measures taken after using components and diverse communications in the IoT field. Three csv files were provided: ‘different_tx’, ‘deep_sleep’, and ‘read_sensor’, where their data was examined as it is mentioned below.

The first step was to process the data as initially all files had a shared structure, with each entry containing a unique field that includes the power data measurement and the corresponding timestamp of recording. In that way, each field was split into 2 different attributes: “Power” and “Timestamp” using Excel. Once the files were updated, the containing data was processed in Python to extract the most relevant information.

Thereafter, timestamps were parsed into Python datetime objects for temporal analysis. Time differences between consecutive entries were computed and cumulatively summed to track the elapsed time. Two new attributes were obtained for each file: 'Difference', which stores the time between samples, and 'Time', which is the cumulative sum of the primer and contains the time elapsed since the beginning of the recording.

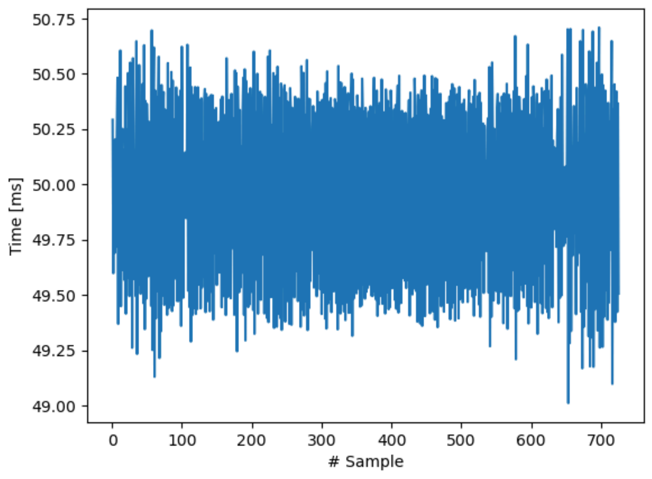

Figure 2. *Time difference between sample’s collection*

This can be done because, as illustrated in *figure 2*, the time between samples is not consistent and varies for every timestamp. That is why, it was considered better to take into account how much time has passed in each moment of the signal.

Consequently, a histogram of power readings was performed to visualize the frequency distribution. The power values were converted from mW to Watts for standardization. In addition, the power readings were segmented into three operational states based on consumption levels: high, medium, and low power. This resulted in a new dataframe for each state, as it is shown in *figure 3*.

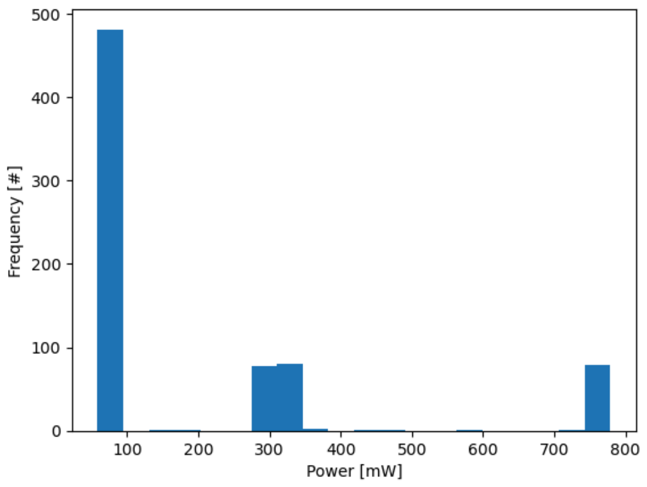

Figure 3. *Power histogram.*

Once the distribution of the data was computed and the different level's data splitted, mean values of each state were calculated. Also, the duration of the device's deep sleep state was calculated, representing one operational cycle. For this, the indices of start and end of each state were identified, and the difference between them was calculated, obtaining the elapsed time, corresponding to the state duration. Multiplying both results, duration in time and power average, as data was reconverted to Watts and Seconds, the equivalent to energy in Joules for each state was computed.

$$Energy\ [J]\ =\ Power\ [W] \cdot Time[s]$$

Time differences between sample collections are graphically represented in *figure 4*. The power evolution over time is visualized, highlighting average power states with horizontal lines.

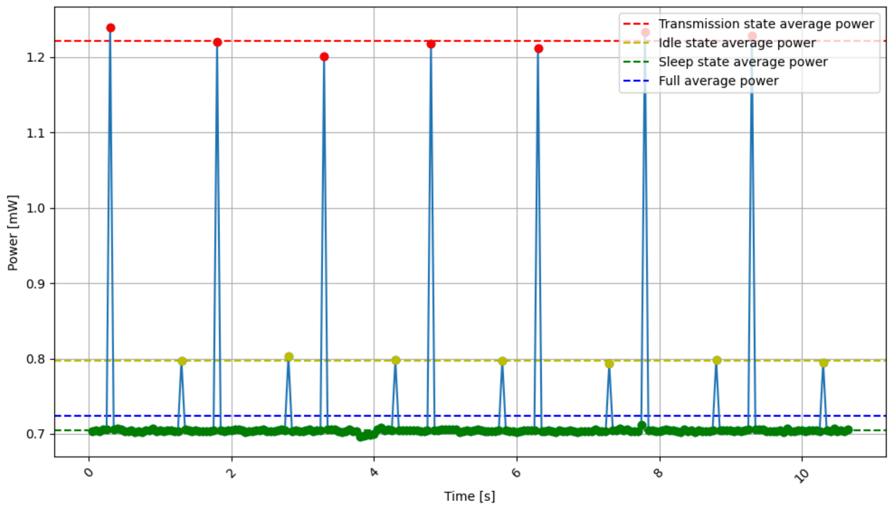

Figure 4. *Power Evolution (different_tx file)*

The complete cycle duration was computed by summing individual state durations in a cycle, which is defined as the time difference between high peaks, and keeps unaltered for the whole signal. Respectively, for the other files, the duration of a cycle was computed as the mean of the duration of all the cycles in a similar way. The total energy consumed per cycle and the number of cycles that the battery can sustain were estimated as follows:

$$Energy\,per\,cycle\,[J] \;= \sum Av.Power_{\;state}\;[W] \cdot TimeDuration[s]$$

$$\#\,cycles\,of\,the\,battery\;= \frac{Energy\,per\,cycle}{Energy\,of\,the\,battery}$$

For our specific case and according to the personal code used, the battery capacity value used for all the calculations is 6140 J. Additionally, the time required to deplete the battery is calculated as follows:

$$Battery\,Lifetime\;= \#\,cycles\,of\,the\,battery\;\cdot\;Cycle\,time\,duration$$

Based on the previous calculations, the results corresponding to the power consumption and the time duration are presented for all the files as it can be observed in the following tables.

Different transmission

The first file, as it is observed in *figure 4*, shows a scenario where the device is used to transmit information. The behavior has a low power standby state and two distinct increased power levels associated with data transmission. The baseline power consumption is steady and minimal, indicating a sleep or low-power idle state. The intermittent spikes represent two different power levels during the transmit phases. The lower transmit mode consumes around 0.8mW, while the higher transmit mode consumes just over 1.2mW. The two levels of power can represent communication with different distanced nodes.

The yellow and red dotted lines at the high power levels represent the average power consumed during the transmission states, while the blue dashed line represents the total average power consumption across all states.

Here are exposed the retrieved values for this file:

	Average Power Consumed [W]	**Average Time Duration [s]**
Transmission High Power	0.061088	0.05
Transmission Low Power	0.039865	0.05
Wifi On	0.061088	1.399513

Table 1. *Average powers obtained for different_tx file states*

Cycle Duration [s]	**Energy used in every cycle [J]**	**Battery # cycles [cycles]**	**Battery lifetime [min]**
1.499513	1.086511	565	141.232

Table 2. *Battery lasting for different_tx file's device*

Deep Sleep

The second file shows a device with three power states: sleep, idle and transmitting. This type of implementation saves power during periods when no transmission is taking place. The sleep state, indicated by the lower green line, shows a power consumption slightly above 0 mW, reflecting a deep sleep or extremely low power state. The yellow dotted line represents the idle state, in which the device remains operational at a moderate power level, shown here at around 300 to 450 mW. The transmission state, marked by the top red dotted line, illustrates periods with the highest power consumption, reaching at around 750 mW.

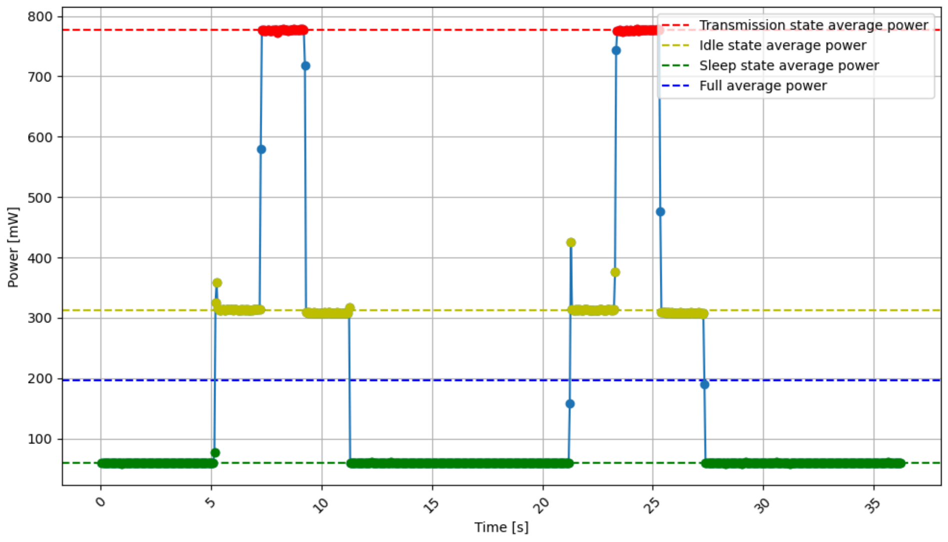

Figure 5. *Power Evolution (deep_sleep file)*

	Average Power Consumed [W]	**Average Time Duration [s]**
Wifi On	1.4750095	1.899266
Idle	1.217804	3.898449
Deep Sleep	0.59033	9.894767

Table 3. *Average powers obtained for deep_sleep file states*

Cycle Duration [s]	**Energy used in every cycle [J]**	**Battery # cycles [cycles]**	**Battery lifetime [hours]**
15.692481	3.2831442	1870	8.15

Table 4. *Battery lasting for deep_sleep file's device*

Read Sensor

The third is related to the file 'sensor_read' and shows the power profile for a sensor that periodically wakes from sleep to read data. The graph shows a continuous low power state with regular intervals of increased power consumption. The green solid line at the bottom shows the sensor in sleep mode, conserving power between reads. The blue spikes represent moments when the sensor is activated to take readings, with power levels rising to around 420 mW. The red dotted line at the top may represent the average power used when the sensor is active and reading data, while the blue dashed line represents the full average power, taking into account both active and sleep states.

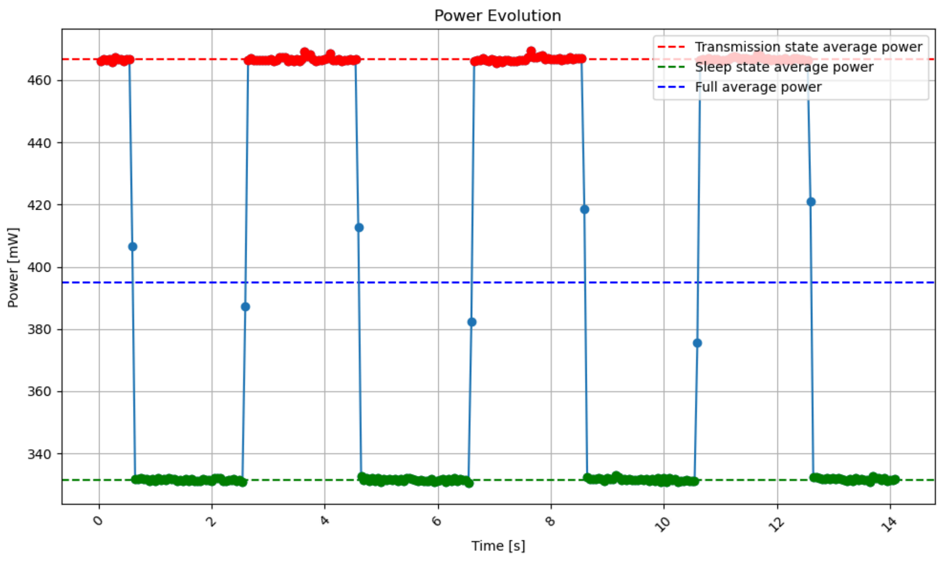

Figure 5. *Power evaluation sensor reading.*

	Average Power Consumed [s]	Average Time Duration [W]
Sensor Reading	1.899114	0.8863695
Idle	1.899208	0.6297503

Table 5. *Average powers obtained for deep_sleep file states*

Cycle Duration [s]	Energy used in every cycle [J]	Battery # cycles [cycles]	Battery lifetime [hours]
1.899114	1.5161198	4049	4.273

Table 6. Table 2. *Battery lasting for read_sensor file's device*

3. Comments Results and Improvements

The graph for the sensor indicates it has periods of low and high power consumption, likely correlating to sleep (low power) and transmission or active sensing (high power) states. However, there are some improvements that could be made to optimize its performance. In one way, for the measurement, it could be implemented adaptive sensing, introducing adaptive sensing intervals where the sensor adjusts its sensing frequency based on certain criteria or environmental triggers, rather than using fixed intervals, for example, if there is a period of time during the day where the parking flux is higher than others.

On the other hand, for the transmission, there can be applied some improvements like data transmission optimization, where compressing data and limiting the transmission of redundant data can reduce the time spent in the high-power transmission state. In that way, batch processing could be more efficient, which means accumulating data and sending it in batches to minimize the number of transmission events. For this specific case this is useful, because if there is a minimum time of occupancy in a parking spot, it is not necessary to send the messages during this period, reducing the transmission periods and thus minimizing the power consumption it generates.

These improvements could lead to significant enhancements in battery life and operational efficiency, making the sensor more suitable for long-term deployment, especially in remote or hard-to-reach locations.