POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Wokwi and Power Consumption

Challenge 1

Author: **Peyman Javaheri Neyestanak**
Iman Javaheri Neyestanak

Student ID:
10976128
10971058
Academic Year: 2024-25

Contents

1 Parking occupancy node specification

For addressing this problem, a parking occupancy node was tested in Wokwi with an ESP32 and ultrasonic distance sensor HC-SR04 to measure the distance in a parking bay, as shown in figure 1. In such a way that if the reading is less than 50 cm then there is a car occupancy, else the parking bay is occupied as free. That message is transmitted to the ESP32 by means of WiFi protocol using esp_now.

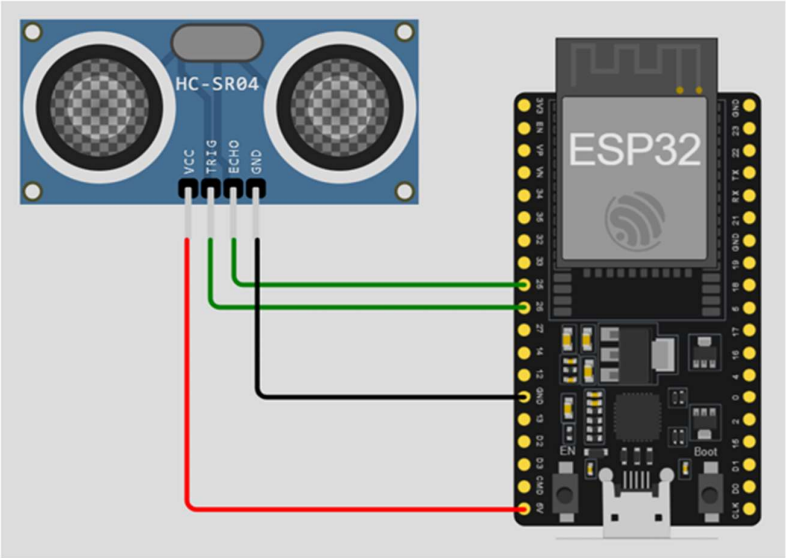

Figure 1: Model

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

$$Distance = pulseMicros/58[cm]$$

Also, to initialize the node, some functions were defined to execute the measurement and the message transmittance. InitializeNetwork() initializes the WiFi first, initializes

the Esp_now protocol, and activates the Peer registration to be used as a reference for 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);
}
```

Besides, distanceMeasure() was utilized to set pin modes, start measurement and read the value. It returns a boolean that accepts true as an argument if the measured distance is greater than 50 cm, i.e., the parking space 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
}
```

On transmission, the esp_now_send() function was used using the broadcast addresses already established and sending the string message of "Free" or "Occupied". The printingTimes() function was also used to print the times based on the sensor measurement and the transmission, as noted 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");
}
```

The value of awake Time is used to determine the time of sleep mode prior to entering sleep mode use. In addition, in this part of the code, with the help of the duty cycle times the us_to_s_factor value, the time for which the sensor is going to sleep is calculated. After this time the deep sleep is initiated.

```
// 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();

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

2 Energy consumption estimation

This section will determine the usage of energy in various steps upon using parts and various communications in the IoT domain. There were three csv files provided: 'Transmission power', 'deep_sleep', and 'read_sensor', whose information was examined as given below.

The first task was to process the data because all files initially had a similar structure, where each record contains a special field that stores the power data measurement and the corresponding timestamp of record.

Next, the timestamps were converted into Python datetime objects for temporal analysis. Two new attributes were obtained for each file: 'Difference', which contained the time difference between samples, and 'Time', the cumulative sum of the primer and incorporating the passage of time from the beginning of recording.

This can be done because the time between samples is not fixed and keeps changing for each timestamp. This is the reason why; it was felt better to consider how much time has passed in each moment of the signal.

Therefore, a histogram of power reading was done to represent the frequency distribution. In addition, the power readings were separated into three operating states based on consumption: high, medium, and low power.

Following the calculation of the data distribution and separating the various level's data, mean values for every state were calculated. Also, the duration of the device's deep sleep state was calculated, which is equivalent to one cycle of operation. For this, the starting and ending indices of every state were calculated, and the difference between them was calculated, obtaining the elapsed time, which is equivalent to the duration of the state. Both results, power average and time duration, were multiplied, as data was converted back to Watts and Seconds, the corresponding energy in Joules for both states was determined.

$$Energy\ [J]\ =\ Power\ [w]\ \times\ time\ [s]$$

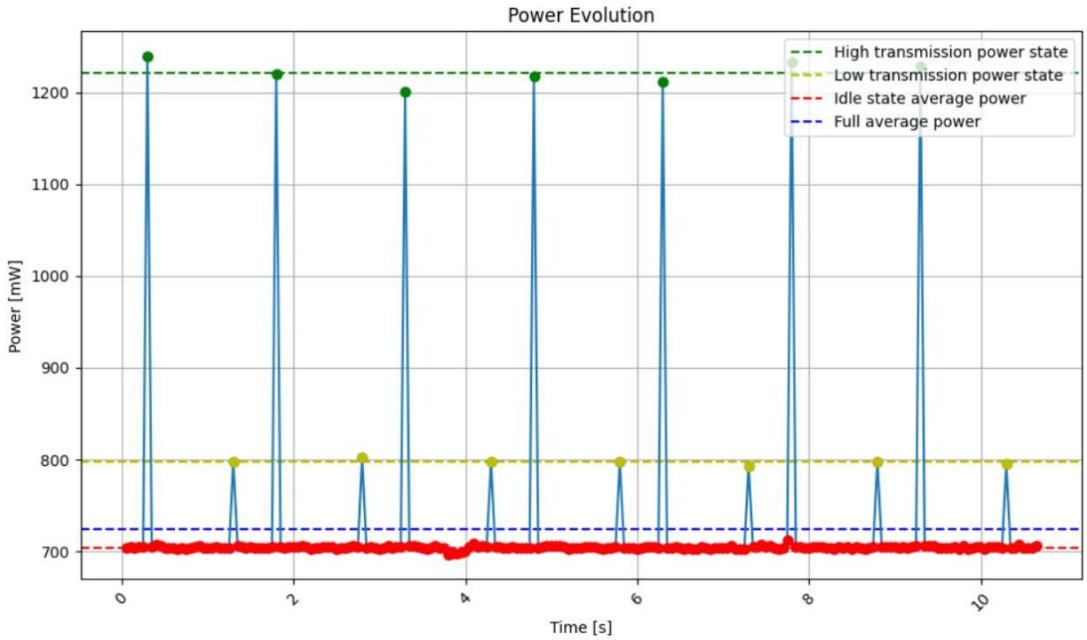

Figure 2: Transmission power

The complete cycle duration was determined by summing up individual state times within a cycle, which is the time difference between high tops and is constant for the whole signal. Similarly, for the rest of the files, the duration of a cycle was determined as the mean of the duration of all the cycles in the same way. The energy used per cycle and the number of cycles the battery can sustain were estimated to be as follows:

$$Energypercycle[J] = \Sigma Avg.Power[W] \times Timeduration[s]$$

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

For our specific case and according to the personal code used, the battery capacity value used for all the calculations is 16128 J. Additionally, the time required to deplete the battery is calculated as follows:

$$Battery\ life\ time\ = \#cycles\ of\ the\ battery\ \times Cycle\ time\ duration$$

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.

2.1. Transmission power

The first file, as it is observed in figure 2, 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 800mW, while the higher transmit mode consumes just over 1200mW. The two levels of power can represent communication with different distanced nodes. The yellow and green 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:

	Avg power [mW]	Energy Consumed [J]	Average time Duration [s]
Transmission High Power	1221.76	0.0611	0.05
Transmission Low Power	797.29	0.0399	0.05
Idle	704.215	0.986	1.3995

Cycle duration	Energy used in every cycle [J]	Battery # cycles [cycles]	Battery lifetime [min]
1.4995	1.0865	14843	370.976

2.2. 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 red 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 green dotted line, illustrates periods with the highest power consumption, reaching at around 750 mW.

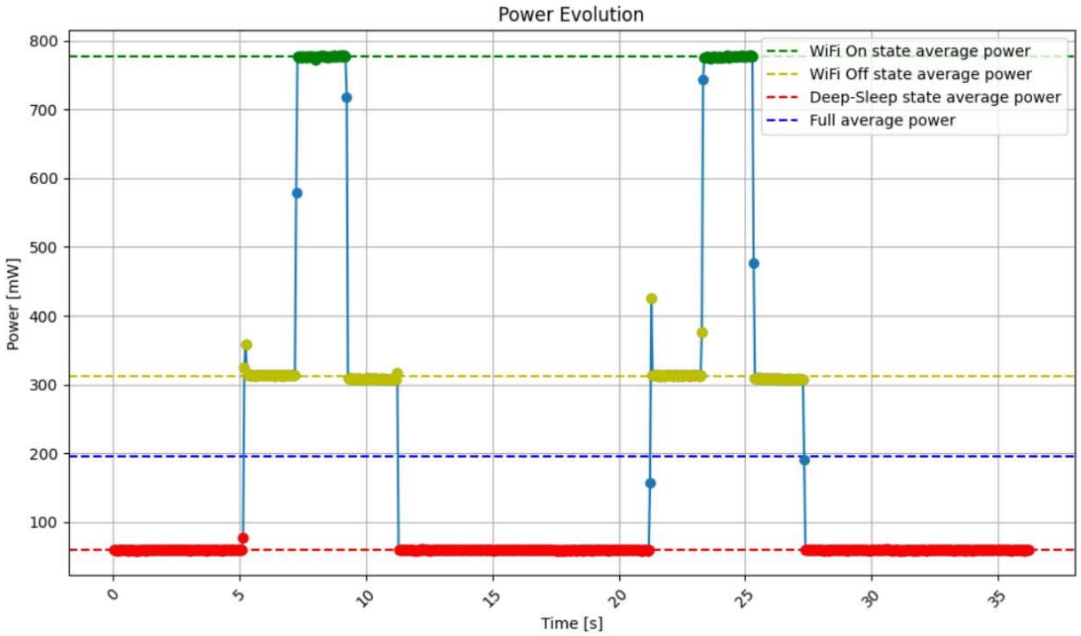

Figure 3: Deep sleep

	Avg Power [mW]	Energy consumed [J]	Average time duration [s]
Wi-Fi On	776.62	1.475	1.899
Wi-Fi Off	312.38	1.226	3.92
Deep sleep	59.66	0.59	9.89

Cycle duration [s]	Energy used in every cycle [J]	Battery # cycles [cycles]	Battery liferime [hours]
15.717	3.291	4900	21.4

2.3. 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 red solid line at the bottom shows the sensor in idle mode, conserving power between reads. The blue spikes represent moments when the sensor is activated to take readings, with power levels rising to around 460 mW. The green 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 idle states.

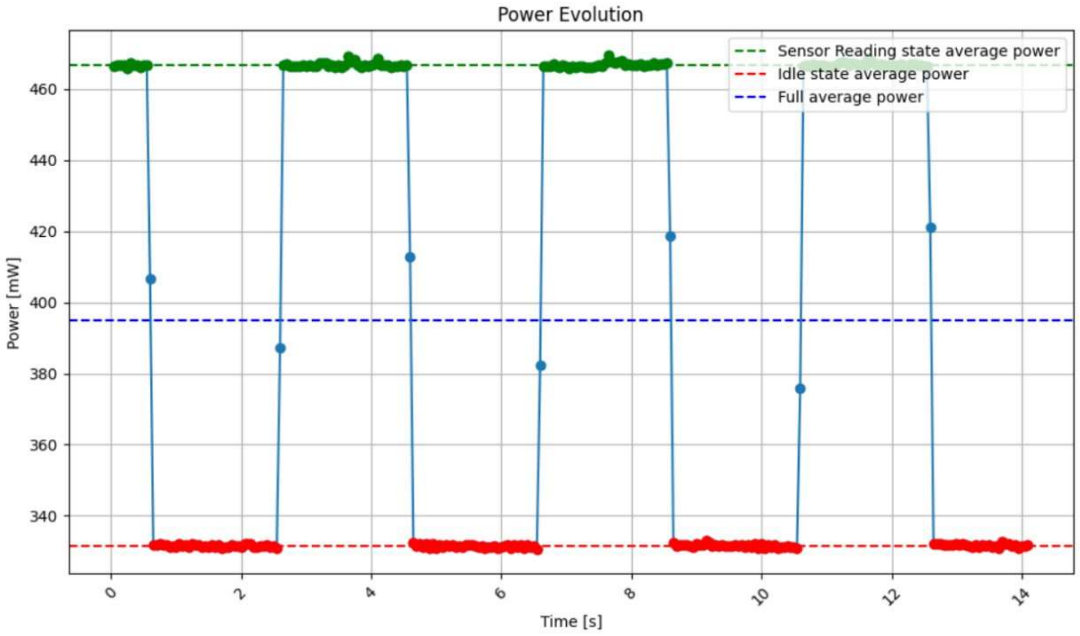

Figure 4: Sensor read

	Avg Power [mW]	Energy Consumed [J]	Avg time duration [s]
Sensor Reading	466.75	0.88637	1.899
Idle	331.59	0.62975	1.899

Cycle duration [s]	Energy used in every cycle [J]	Battery # cycles [cycles]	Battery lifetime [hours]
3.798	1.51612	10637	11.22

3 Comment on Results and Improvements

The power consumption chart of the sensor indicates low and high consumption periods, which necessarily reflect low-power (sleep) and high-power periods (transmission or sensing active). Still, certain things can be done to enhance its performance optimally. In a way, to the measurement, adaptive sensing could be applied, with adaptive sensing intervals where the sensor will adjust its sensing rate based on some conditions or environmental triggers, rather than fixed intervals, for example, if there is a particular time of the day when the parking flux is higher than others.

On the other hand, for transmission, certain improvements like optimization of data transmission can be applied, where data compression and avoidance in the transmission of redundant data can reduce the duration in the high-power transmission mode. In this way, batch processing could prove to be better, i.e., the gathering of data and its transmission in batches to cut down on the frequency of transmission events. This is useful for this specific case, as if there is a minimum time occupancy in a parking space, it does not need to send the messages within this time, reducing the transmission times and thus the power consumption it generates. These improvements would lead to tangible gains in battery life and operating efficiency, enabling the sensor to be more effectively deployed for longer periods, especially in remote or inaccessible locations.

4 Optimizing sink position in a Wireless Sensor Network

4.1. Lifetime of the system

To calculate the lifetime of the system, first we should calculate the distance from each sensor to the sink, then the energy consumption for each sensor. Finally, we compute the lifetime of each sensor:

```python
import numpy as np

# Given data
sink_position = (20, 20)
sensor_positions = [(1, 2), (10, 3), (4, 8), (15, 7), (6, 1), (9, 12), (14, 4), (3, 10), (7, 7), (12, 14)]
initial_energy = 5  # mJ
packet_size = 2000  # bits
Ec = 50e-9  # nJ/bit converted to J/bit
k = 1e-9  # nJ/bit/m² converted to J/bit/m²

# Compute distances and energy consumption per transmission
lifetimes = []
for (x, y) in sensor_positions:
    d = np.sqrt((sink_position[0] - x)**2 + (sink_position[1] - y)**2)  # Euclidean distance
    E_tx = packet_size * (Ec + k * d**2)  # Energy per transmission in J
    E_tx_mJ = E_tx * 1e3  # Convert to mJ
    T_i = (initial_energy / E_tx_mJ) * 10  # Lifetime in minutes
    lifetimes.append(T_i)

# Find system lifetime
system_lifetime = min(lifetimes)
system_lifetime
```

And the system lifetime is approximately 34.01 minutes.

4.2. Optimal position of the sink

To find the optimal position of the sink that maximizes the system lifetime, we need to minimize the energy consumption of the sensor consuming the most energy.

```
from scipy.optimize import minimize

# Define the worst-case energy consumption function
def max_energy_consumption(sink_pos):
    xs, ys = sink_pos
    max_energy = 0
    for (x, y) in sensor_positions:
        d = np.sqrt((xs - x)**2 + (ys - y)**2)  # Euclidean distance
        E_tx = packet_size * (Ec + k * d**2)  # Energy per transmission in J
        E_tx_mJ = E_tx * 1e3  # Convert to mJ
        max_energy = max(max_energy, E_tx_mJ)  # Track worst-case sensor energy
    return max_energy

# Initial guess for sink position (center of sensor area)
x0 = np.mean([s[0] for s in sensor_positions])
y0 = np.mean([s[1] for s in sensor_positions])

# Perform optimization
result = minimize(max_energy_consumption, x0=(x0, y0), bounds=[(0, 20), (0, 20)])

# Optimal sink position
optimal_sink_position = result.x
optimal_sink_position
```

```
array([6.90470949, 7.6290163 ])
```

As we can see, the optimal sink position is approximately [6.9 , 7.6].

4.3. Trade-offs

The advantages of fixed sink position are lower complexity, predictable routing and lower hardware cost, while the disadvantages are energy imbalances, reduced system lifetime and bottlenecks.

On the other hand, the pros of dynamically moving sink are balanced energy usage, extended system lifetime and reduced transmission costs, while the cons are higher complexity, increased overhead and potential delays.