IoT: Challenge1

Ni Giovanni 10831328 (LEADER)

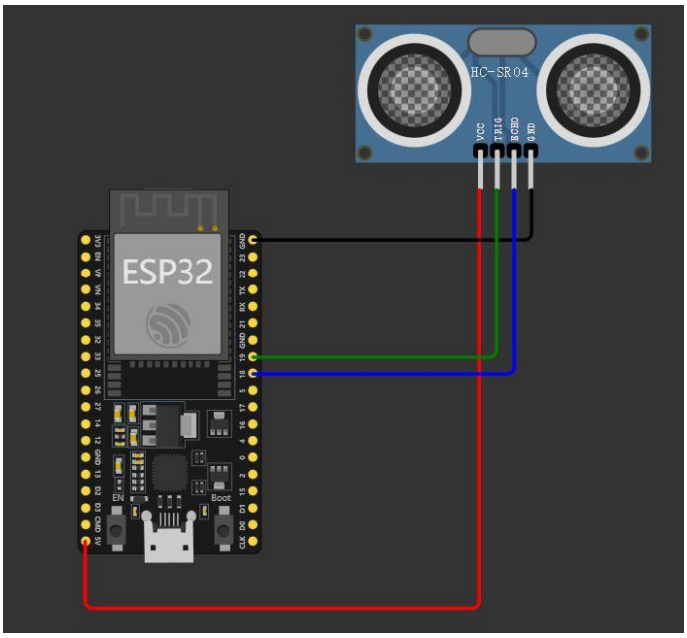
Gu Xinyue 10840236

Wowki project public link:

https://wokwi.com/projects/425898051562761217

1. ***Develop on Wokwi a simple parking occupancy node using the HC-SR04 ultrasonic distance sensor and the ESP-NOW for communication.***
   Go in Deep Sleep with a defined duty-cycle X.

First, we start with the hardware part. We analyze how HC-SR04 ultrasonic distance sensor works, and how we should connect to the ESP board.



HC-SR04 has four pins, we connect VCC to 5V, GND to Groud, TRIG to pin 19 and ECHO to pin 18.

The HC-SR04 measures distance based on the time of flight of the ultrasonic wave. The standard formula for calculating the distance is:

$$Distance = \frac{EchoTime * SoundSpeed}{2}$$

In air, the speed of sound is approximately 343 m/s (or 0.0343 cm/µs), so the calculation would be: distance = duration × 0.0343 / 2, which gives the result in centimeters (cm).
Our duty-cycle X is 28%50 + 5 = 33 s

looking at the code:

```
// IoT Challenge1: Ni Giovanni, Gu Xinyue
#include <WiFi.h>
#include <esp_now.h>
#include <esp_wifi.h>

#define PIN_TRIG 19 //connecting to the pin 19
#define PIN_ECHO 18 // connecting to the pin 18
#define US_TO_S 1000000 //constant used in the function esp_sleep_enable_timer_wakeup()
#define SOUND_SPEED 0.034
#define DUTYCYCLE 33 //for our group: 28 % 50 + 5

RTC_DATA_ATTR String status = "free";

// MAC receiver: we need to test it in a simulationn
// so in this case we choose the BROADCAST: 8C:AA:B5:84:FB:90.
uint8_t broadcastAddress[] = {0x8C, 0xAA, 0xB5, 0x84, 0xFB, 0x90};

esp_now_peer_info_t peerInfo;
```

We start from including all library we need: wifi.h for wifi connection, esp_now.h for the protocol, esp_wifi.h is a library that we have included to test some parameters, in particular in line 91,92,93 (commented)to test the transmission power.
Then define some constant value from line 6 to line 10 and define the broadcast address, finally declare the variable “status”. We use the attribute RTC_DATA_ATTR for the variable status because the RTC_DATA_ATTR attribute in ESP32 is used to store variables in the RTC (Real-Time Clock) memory. This memory is retained even when the ESP32 enters deep sleep mode, allowing variables to persist across deep sleep cycles.

To make the code clearer, we decided to put code in functions, and call the function every time we want to use it.

```
36  void setup_WiFi() {
37    WiFi.mode(WIFI_STA);
38    esp_now_init();
39    //send callback
40    esp_now_register_send_cb(OnDataSent);
41    //receive callback
42    esp_now_register_recv_cb(OnDataRecv);
43
44    // Peer Registration
45    memcpy(peerInfo.peer_addr, broadcastAddress, 6);
46    peerInfo.channel = 0;
47    peerInfo.encrypt = false;
48    // Add peer
49    esp_now_add_peer(&peerInfo);
50  }
```

Function void setup_Wifi(), there are all codes we need for wifi connection and esp_now protocol.

```
21  // Sending callback
22  void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
23    Serial.print("Send Status: ");
24    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Ok" : "Error");
25  }
26
27  //Receiving Callback
28  void OnDataRecv(const uint8_t* mac, const uint8_t *data, int len) {
29    Serial.print("Message received: ");
30    char receivedString[len];
31    memcpy(receivedString, data, len);
32    Serial.println(String(receivedString));
33  }
```

OnDataSent() and OnDataRecv() are used for the esp_now protocol, describing the behavior of when the board is sending or receiving some information.

```
52  //measurement process: HC-SRO4 will return the status depending on the distance.
53  String measurement(){
54    String message;
55    // Start a new measurement:
56    digitalWrite(PIN_TRIG, HIGH);
57    delayMicroseconds(10);
58    digitalWrite(PIN_TRIG, LOW);
59
60    // Read the result:
61    int duration = pulseIn(PIN_ECHO, HIGH);
62    int distance = duration*SOUND_SPEED /2; //formula
63    Serial.print("Distance in CM: ");
64    Serial.println(distance);
65
66    if(distance<=50) {
67      message = "occupied";
68    } else {
69      message = "free";
70    }
71    return message;
72  }
```

A measurement function that measures the distance with the sensor, as we said before the sensor measures the distance based on the time of flight. So (line 61) we have first to read the duration, then with the formula (line 62) calculate the distance in cm. This function not only does the measurement but returns the status of the parking spot: “occupied” if the distance is <= 50cm, “free” otherwise.

```
74  void setup() {
75    unsigned long start = millis(); //take the starting time
76
77    Serial.begin(115200);
78    pinMode(PIN_TRIG, OUTPUT);
79    pinMode(PIN_ECHO, INPUT);
80
81    setup_WiFi();
82
83    status = measurement();
84
85    esp_now_send(broadcastAddress, (uint8_t*)status.c_str(), status.length() + 1);
86    //if necessary, we need to wait the esp finish sending
87    //delay(100);
88
89    //to measure the current tx power
90    //int8_t txPower;
91    //esp_wifi_get_max_tx_power(&txPower);
92    //Serial.println("current tx power: ");
93    //Serial.println(txPower);
94
95    unsigned long end = millis(); //take the ending time
96    Serial.println(String(end-start));  //record
97
98    //deep sleep
99    esp_sleep_enable_timer_wakeup(DUTYCYCLE * US_TO_S);
100   Serial.println("ESP 32 sleep 33 s duty cycle");
101   Serial.flush();
102   esp_deep_sleep_start();
103 }
```

This is our setup() function, where we “merge” codes. We use millis() to take note of different times, for example process starting and ending time, because we will use this data in the second part of the challenge. We setup the pin mode at line 78 and 79, setup the wifi at line 81, obtained the parking spot status at line 83, and we send it for broadcast at line 85 using the function esp_now_send(). From line 99 to 102 is the deep sleep part.

Some examples of outputs:

- Parking spot occupied:

```
load:0x40080400,len:2972
entry 0x400805dc
Distance in CM: 45
Send Status: Ok
Message received: occupied
ESP 32 sleep 33 s duty cycle
ets Jul 29 2019 12:21:46
```

Since the distance is 45 cm <= 50cm, the system sends a message: occupied. Send Status is ok, and the message is correctly received.

- Parking spot free:

```
load:0x40080400,len:2972
entry 0x400805dc
Distance in CM: 277
Send Status: Ok
Message received: free
ESP 32 sleep 33 s duty cycle
ets Jul 29 2019 12:21:46
```

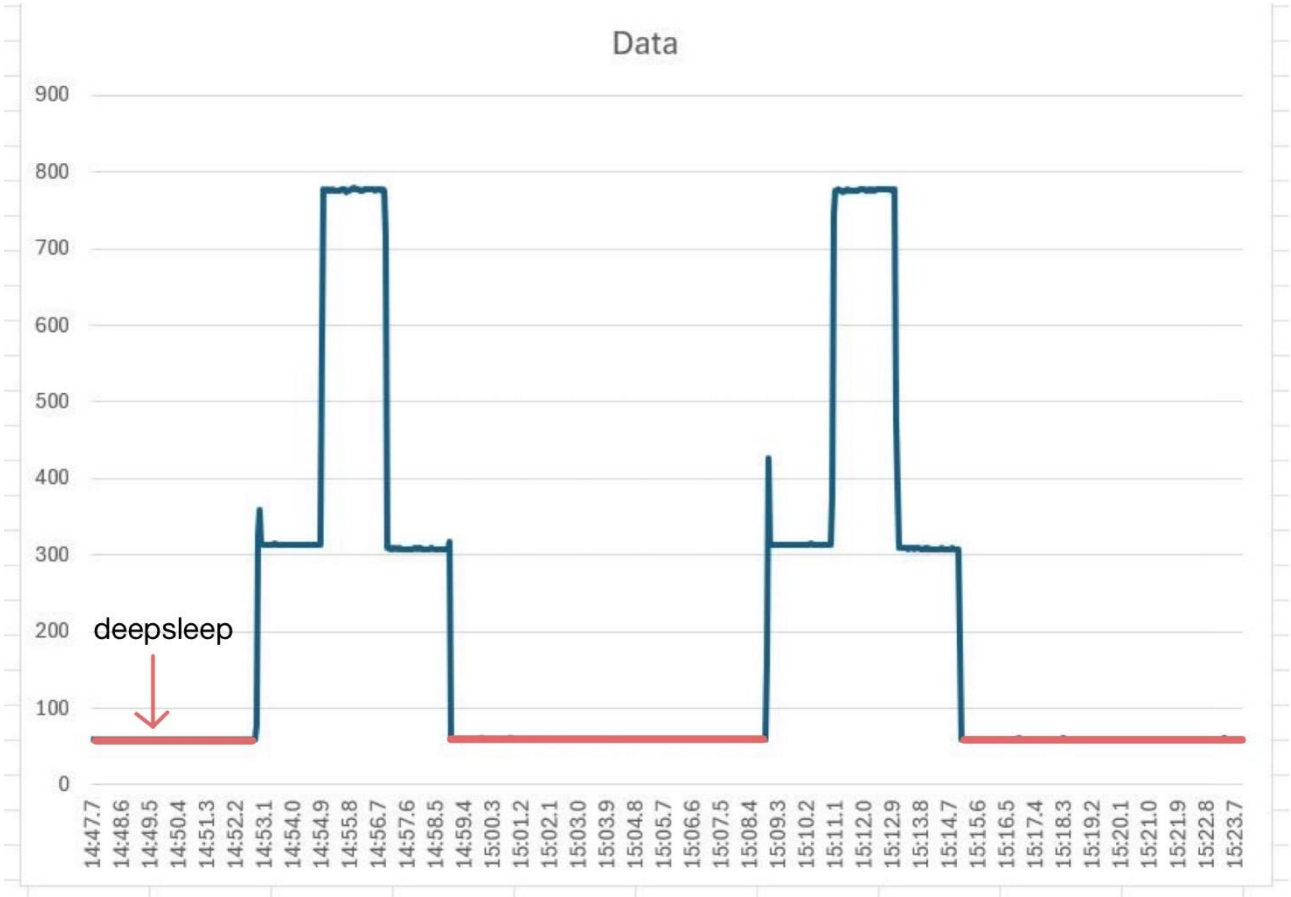
Like the previous example, but in this case the distance 277cm > 50cm so the system sends the free message.

2. Compute the duty-cycle from point one and perform an Energy Consumption estimation of the sensor node:

Question1:
Estimate the average Power consumption for each state of the node (Deep Sleep state, Idle, Transmission State, Sensor reading).

From the CSV file provided on WeBeep, we obtained three different graphs:

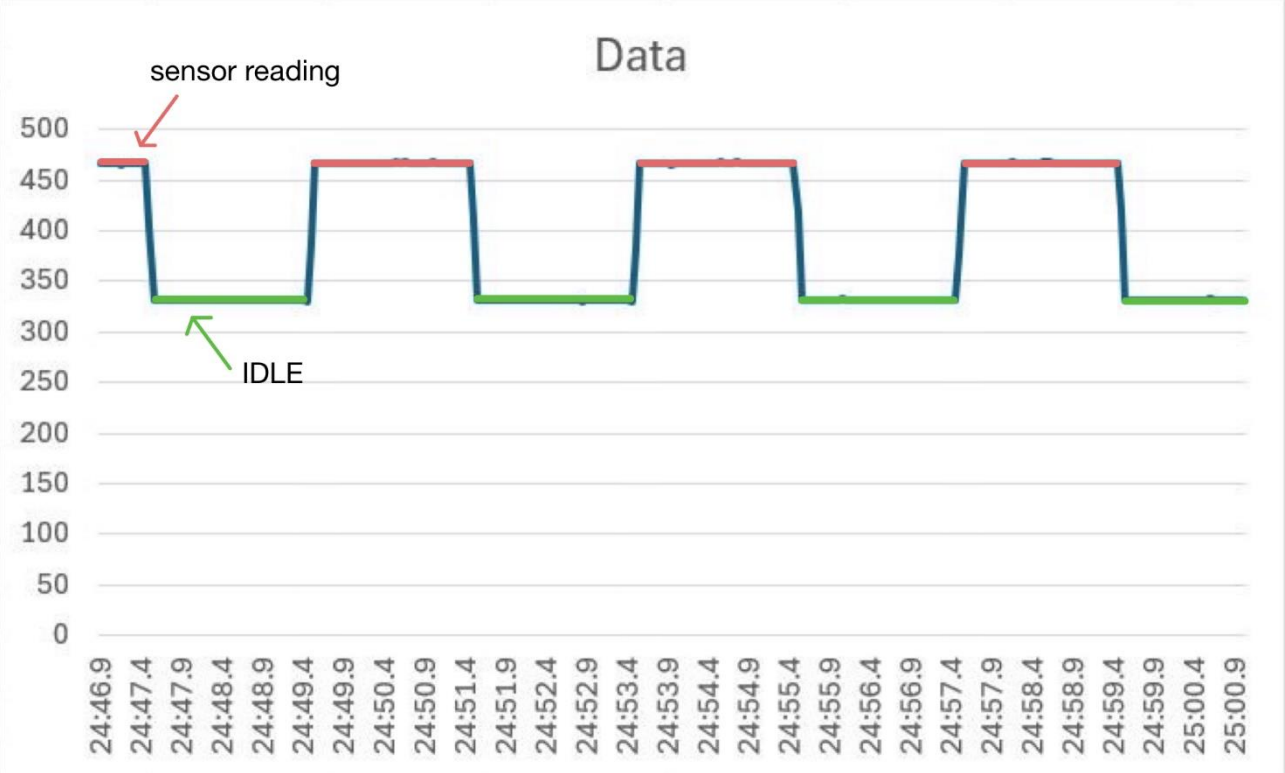
deep_sleep.csv



We observe that there are three periods of time in deep_sleep state, with an average value of:

$$P_{DeepSleep} = 59.62mW$$

Sensor_read.csv

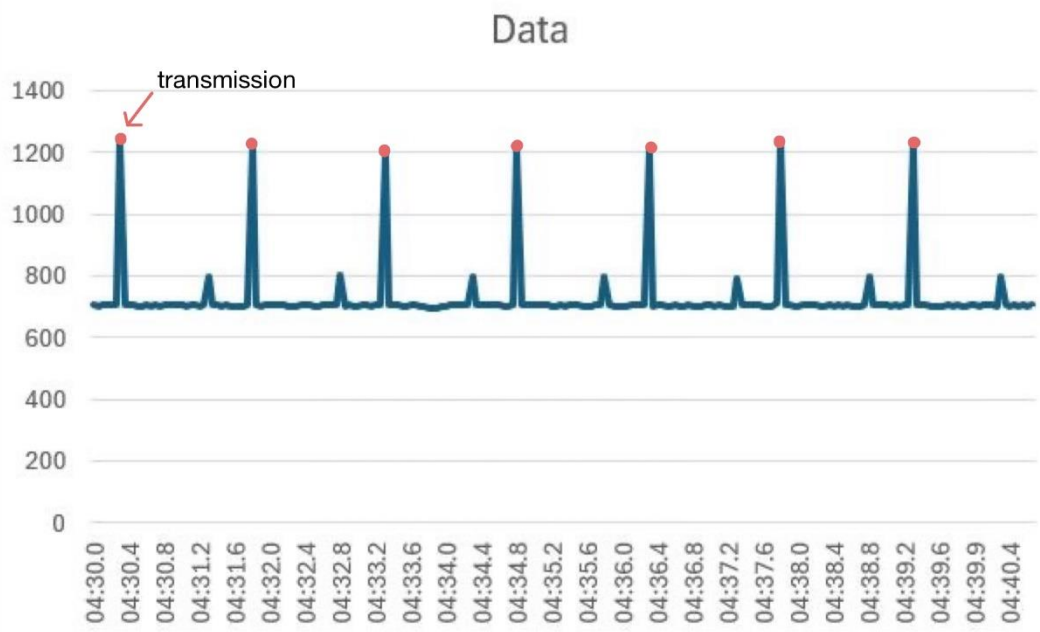


Here we have two different states: Idle and sensor reading.
From the datasets, we obtained that in average:

$$P_{Idle} = 331.59mW$$

$$P_{SensorReading} = 466.75mW$$

Trasmission_power.csv



Here we have the transmission state, we consider that with 19dBm.
In average

$$P_{trasmission} = 1221.76mW$$

Question2: Estimate the Energy consumption of one transmission cycle.

As we said in the first part: our deep sleep time is X = 33 seconds.

For the entire cycle time, we add in different part of codes the function millis() to take note of precise time we are interested in.

```
unsigned long start = millis(); //take the starting time
```

```
unsigned long end = millis(); //take the ending time
Serial.println("time: " + String(end-start));  //record
```

In this way we obtained the following output:

```
entry 0x400805dc
Distance in CM: 46
time: 186
ESP 32 sleep 33 s duty cycle
```

So, approximately 0.19s

Now, our energy consumption in Deep Sleep state is:

$$E1: E_{DeepSleep} = P_{DeepSleep} * 33 = 59.62mW * 33s = 1967.46mJ.$$

The energy consumption in the remaining states is:
E2: $E_{Idle+Transmission+SensorReading}$ = (331.59+ 1221.76+ 466.75) mW* 0.19s = 383.82mJ

The energy consumption of 1 Transmission cycle is the sum of them.
E1 + E2 = 1967.46mJ + 383.82mJ = 2351.28mJ approximately 2.35J.

Question3: Estimate the time the sensor node lasts before changing the battery.

In our case

- the Energy of the battery is Y= 1328%5000 + 15000 = 16328J
- $T_{transmission}$ = 33s + 0.19s = 33.19s
- $E_{consumption}$ = 2.35J

So:

$$Battery_{life} = \frac{E_{battery}}{E_{consumption}} * T_{transmission} = \frac{16328J}{2.35J} * 33.19s = 230606s = 64hours = 2.6days$$

3. Comment Result and Improvements:

Summary and comment:

The ESP32 microcontroller is set up to monitor the availability of a parking space using an ultrasonic sensor. It sends the status of the space to a receiver node via ESP-NOW communication. After sending this information, the microcontroller enters deep sleep mode to conserve energy until it's time to wake up again.

The parking space status is stored in the RTC (Real-Time Clock) memory using the RTC_DATA_ATTR attribute, ensuring that the status remains intact even when the device goes into deep sleep.

To check if the parking space is occupied, an ultrasonic sensor is used. It emits sound waves and measures how long it takes for the waves to reflect back, calculating the distance to any object in the way. This distance is used to determine whether the space is occupied or not.

For power conservation, after the status check and data transmission, the microcontroller enters deep sleep mode (esp_deep_sleep_start()). In this mode, most components of the device are turned off, but the RTC memory remains active, ensuring that the device can wake up at the correct time. This helps extend the battery life during periods of inactivity.

The function esp_sleep_enable_timer_wakeup() is used to set a timer-based wake-up schedule. The device will automatically wake up after a set amount of time (defined by TIME_TO_SLEEP), allowing periodic checks of the parking space status without requiring the device to stay active all the time.

Possible improvements:

From our calculation the battery should be replaced after 2.6 days, we can do better.

- One enhancement could be for the ESP device to check the parking space's occupancy status right after waking from deep sleep mode. If the status has not changed (since we stored it in the RTC memory, which persists through deep sleep), the device could go back to sleep without turning on Wi-Fi or sending a message to the receiver. This would help conserve energy while keeping track of parking spaces efficiently.

- Another way to improve is instead of a fixed X-second interval, use an adaptive approach based on historical data or real-time activity trends. For example, during peak hours, the node can wake up more frequently, but at night, it can extend the sleep interval. Obviously, the peak hours depend on where the sensor is physically, for example the parking spot of a supermarket is usually free during the night.

- A small improvement could also be done by sending compressed messages, for example 0 for FREE status and 1 for OCCUPIED status, instead of full words.