Internet Of Things (IOT) - Challenge 1

Mohammadreza Zamani (10869960) – Asal Abbasnejadfard (10974178)

Wokwi and Power Consumption

1. Parking occupancy node specification

Introduction:

This project focuses on developing an energy-efficient sensor node for monitoring parking occupancy using an HC-SR04 ultrasonic distance sensor, as it is shown in Fig. 1. The node detects whether a parking slot is free or occupied based on the measured distance and transmits this status to a central ESP32 sink node via the ESP-NOW protocol. link to the project (https://wokwi.com/projects/425241064895099905).

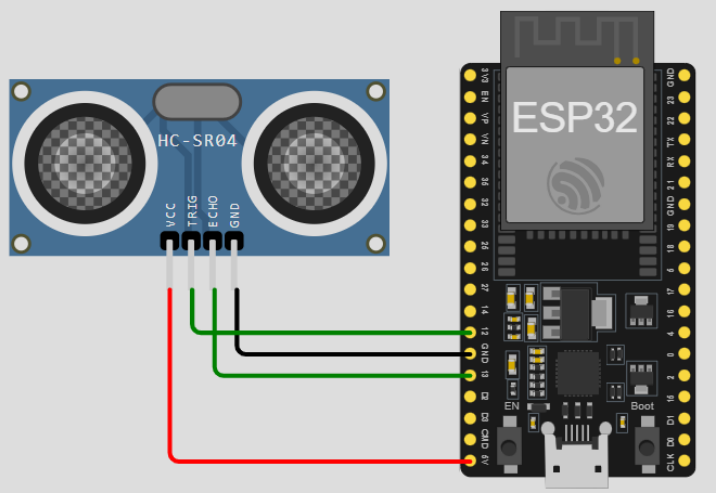

Fig.1. Node structure.

The HC-SR04 has input pins VCC (5V) for power and TRIG to send ultrasonic pulses and an output pin ECHO, which returns the duration of the received pulse. The sensor calculates the distance based on the time difference between sending and receiving the pulse, using the speed of sound.

Implementation:

In this section, we will explain the main parts of the code. A few functions were defined for the node's development in order to accomplish message measurement and transmission.

```
// Measure distance using the HC-SR04
float getDistance() {
    digitalWrite(TRIGGER_PIN, LOW);
    delayMicroseconds(2);  // Delay for stability
    digitalWrite(TRIGGER_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIGGER_PIN, LOW);
    long duration = pulseIn(ECHO_PIN, HIGH);
    return (duration * 0.0343) / 2;  // Convert time to distance in cm
}
```

The TRIGGER_PIN is set LOW for 2 microseconds for stability. Then, a 10-microsecond pulse is sent to trigger the ultrasonic sensor. pulseIn(ECHO_PIN, HIGH) measures the time taken for the sound wave to return.

```
// ESP-NOW callback function for transmission completion
void onDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
    Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Success" : "Fail");
    transmissionDone = true;  // Mark transmission as complete
}
```

This callback function executes when data is sent via ESP-NOW. If the message is successfully sent, it prints "Success"; otherwise, it prints "Fail."

```
// Send parking status using ESP-NOW
void sendStatus(const char* status) {
    transmissionDone = false;
    unsigned long transmissionStartTime = millis();  // Start transmission timer
    esp_err_t result = esp_now_send(broadcastAddress, (uint8_t*)status, strlen(status) + 1);

    if (result == ESP_OK) {
        Serial.println("Message sent successfully");
    } else {
        Serial.println("Error sending message");
    }

    // Wait for the callback to confirm transmission
    unsigned long waitStart = millis();
    while (!transmissionDone && millis() - waitStart < 100) {  // Timeout after 100ms
        delay(1);
    }

    unsigned long transmissionTime = millis() - transmissionStartTime;  // Calculate actual transmission time
    Serial.println("Transmission Time: " + String(transmissionTime) + " ms");
}
```

The sendStatus function is responsible for sending the parking occupancy status ("OCCUPIED" or "FREE") to the ESP32 sink node using ESP-NOW. It ensures that the message is transmitted successfully and measures the transmission time. The function waits until the onDataSent callback is triggered, indicating that the transmission was completed. If the transmission takes too long (> 100 ms), it times out and continues execution.

```
void onDataRecv(const esp_now_recv_info_t *info, const uint8_t *data, int len) {
    Serial.print("Received Parking Status: ");
    Serial.println((char*)data);
}
```

This function executes when data is received via ESP-NOW. The received parking status is printed to the serial monitor.

```
void setup() {
    unsigned long startMillis = millis();  // Time of booting

    Serial.begin(115200);
    Serial.println("ESP32 Wake-Up Time: " + String(millis()) + " ms");  // Log wake-up time

    WiFi.mode(WIFI_STA);
    pinMode(TRIGGER_PIN, OUTPUT);
    pinMode(ECHO_PIN, INPUT);

    if (esp_now_init() != ESP_OK) {
        Serial.println("ESP-NOW initialization failed!");
        return;
    }

    esp_now_register_recv_cb(onDataRecv);
    esp_now_register_send_cb(onDataSent);  // Register callback for send completion

    memcpy(peerInfo.peer_addr, broadcastAddress, 6);
    peerInfo.channel = 0;
    peerInfo.encrypt = false;
    if (esp_now_add_peer(&peerInfo) != ESP_OK) {
        Serial.println("Failed to add peer");
        return;
    }
```

The setup function initializes the ESP32, configures the Wi-Fi and ESP-NOW communication, measures the parking slot status, transmits the data, and then enters deep sleep to save power. Let's go through it in a smooth, logical order. The ESP-NOW is initialized for wireless communication. If initialization fails, it prints an error message and stops execution. The onDataRecv handles incoming messages, and onDataSent confirms if the message was sent successfully.

```
Serial.println("Enabling Wi-Fi 2sec...");
unsigned long wifiEnableTime = millis();  // Time Wi-Fi turned on
delay(2000);

// Measure distance
unsigned long sensorStartTime = millis();  // Start measuring time
float distance = getDistance();
unsigned long sensorReadingTime = millis() - sensorStartTime;  // Time spent on sensor reading

Serial.print("Measured Distance: ");
Serial.print(distance);
Serial.println(" cm");

const char* parkingStatus = (distance <= THRESHOLD_DISTANCE) ? "OCCUPIED" : "FREE";  // Determine parking status
Serial.print("Parking Status: ");
Serial.println(parkingStatus);

// Send data
sendStatus(parkingStatus);
Serial.println("Disabling Wi-Fi 2sec...");
WiFi.mode(WIFI_OFF);
unsigned long wifiDisableTime = millis();  // Time Wi-Fi turned off
delay(2000);
```

Wi-Fi remained enabled for 2 seconds to ensure ESP-NOW communication is stable before sending data. getDistance () is called to measure the distance of the object in front of the sensor. If the distance is ≤ 50 cm, the parking slot is marked as "OCCUPIED". Otherwise, it is marked as "FREE". Wi-Fi is turned off to save power.

2. Energy Consumption Estimation

The purpose of this part is to determine the energy consumption of different actions performed following the use of various IoT components. The data from three CSV files—"transmission_power," "deep_sleep," and "read_sensor"—was analyzed. In the first step, all data was processed. Each row in the data includes the power measurement and the corresponding timestamp. It was thought to be better to assess the amount of time that has passed in each signal moment because the time between samples is inconsistent.
One operational cycle was calculated. To determine the elapsed time, which corresponds to the state duration, the start and end indices of each state were determined, and the difference between them was computed. The energy is computed for each state according to the below equation.

$$E\,[J] \;=\; P\,[W] \cdot T[s]$$

The duration of each state must be known to calculate energy consumption during one cycle. This information was obtained by using Wokwi.

- Deep sleep time: 9.634 (s)
- Idle (booting) time: 1.369 (s)
- Wifi turn-on time: 5.586 (s)
- Wifi turn-off time: 7.623 (s)
- Sensor reading time: 0.017 (s)
- Transmission time: 0.001 (s)
- Cycle Duration: 9.634 + 1.369+ 5.586+ 7.623+ 0.017+ 0.001 = 24.23 (s)

As we know,

- Deep sleep power consumption: 59.62 mW
- Idle power consumption: 322.5 mW
- Wifi turn-on power consumption: 740.42 mW
- Wifi turn-off power consumption: 308.27 mW
- Sensor reading power consumption: 466.74 mW
- Transmission power consumption: 797.29 mW

We use the equations below to calculate energy consumed per cycle and the number of cycles of batteries and battery lifetime.

$$E\ [Joules] = \sum \text{Average power [m}Watt] \cdot \text{Time [Seconds]}$$

Number of battery cycles = Battery Energy / E [Joules]

Battery lifetime = Number of battery cycles . time duration of cycle

Energy consumption of 1 transmission cycle is:

E_{cycle} = (9.634) (59.62) + (1.369) (322.5) + (5.586) (740.42) + (7.623) (308.27) + (0.017) (466.74) + (0.001) (797.29) = 7510.541 mJ

Number of battery cycles = 19960 / 7.51 = 2658 cycles

Battery lifetime = 2658 . 24.23 = 64403.34 (s)

Transmission Power:

The power consumption is related to transmission_power.csv on overtime is displayed on Fig.2. About 0.8 mW is used in the lower transmit mode and slightly more than 1.2 mW in the higher transmit mode. In Fig.2 the green and yellow dotted lines show the Transmission average power and idle average power transmission, respectively. Also, the black dotted line displays the full average power consumption. According to challenge 1 pdf, the energy of battery in our case is 19960 joules.

The power data were separated into three states—high, medium, and low power consumption—for easier result analysis.

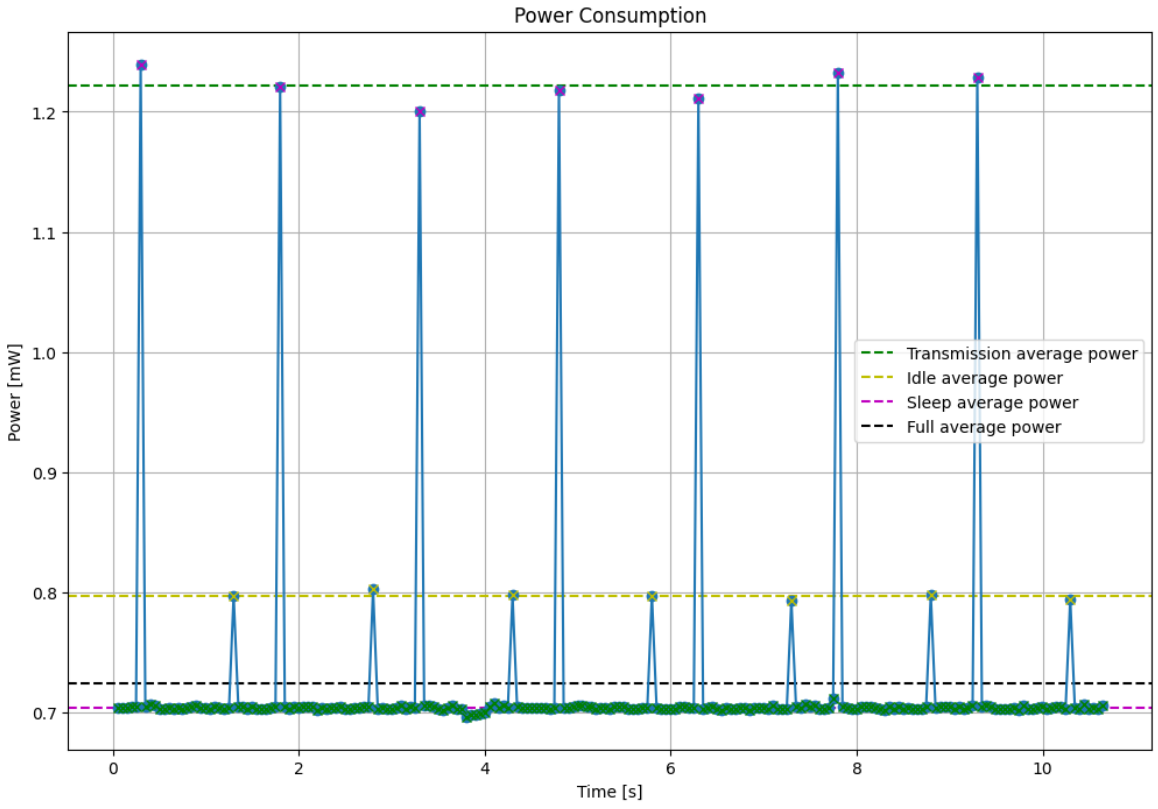

Fig. 2. Power Consumption (transmission_power.csv).

The duration of a cycle was calculated as the average duration of all the cycles for every CSV file.

The results of this part:

Table. 1

	Avg. Power Consumed [W]	Avg.Time Duration [s]
High Power Transmission	0.06108	0.05
Medium Power Transmission	0.03986	0.05
Low Power Transmission	0.06108	1.39951

Table. 2

Duration of cycle [s]	Energy used in cycle [J]	Number of cycles of battery	Battery lifetime [min]
1.499512	1.086511	18370	459.119

Deep Sleep:

The lower blue line, which represents the sleep state, indicates deep sleep with a power usage just above 0 mW. The top green dotted line, which represents the transmission condition, shows times when power consumption is at its peak, which is approximately 750 mW.

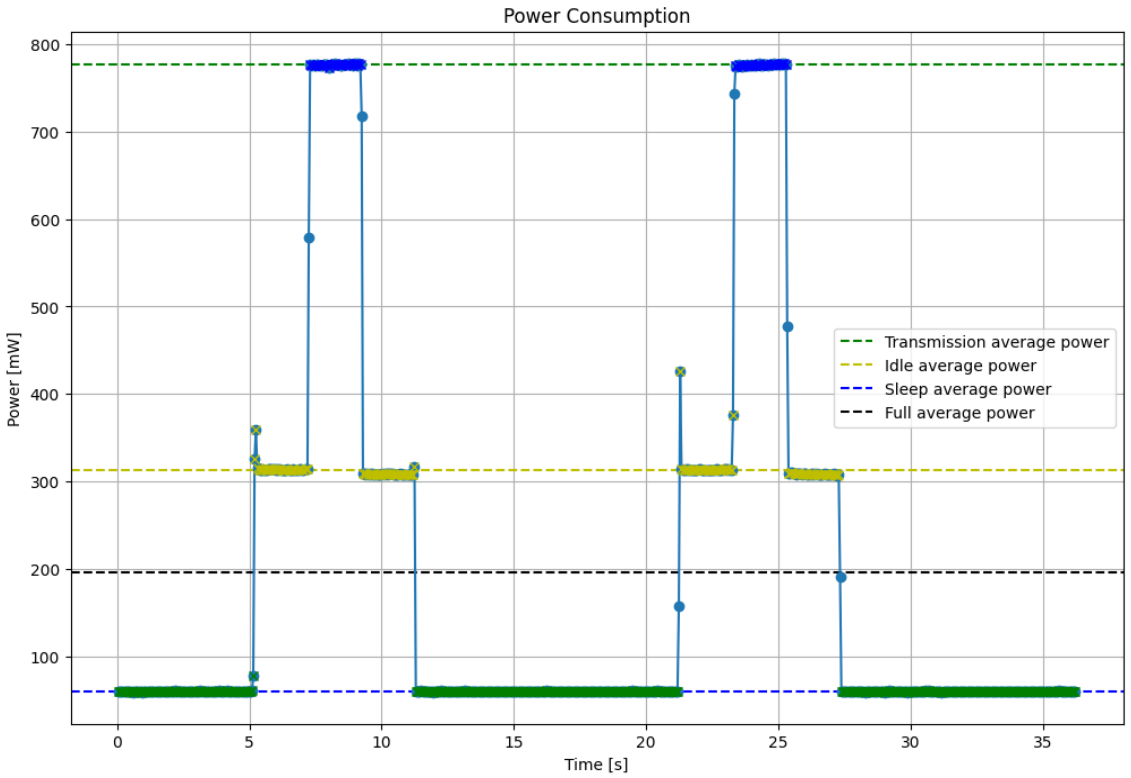

Fig.3. Power Consumption (deep_sleep.csv).

Table. 3

	Average Power Consumed [W]	Average Time Duration [s]
Wi-Fi On	1.475009	1.89926
Idle	1.21780	3.89844
Deep Sleep	0.5903	9.89476

Table. 4

Duration of the cycle [s]	Energy used in cycle [J]	Number of cycles of battery	Battery lifetime [hours]
15.6924	1.808136	11038	48.119

Read Sensor:

The graph displays a persistently low power condition interspersed with regular spikes in power usage. The sensor is in sleep mode as indicated by the green solid line at the bottom. The average power consumption when the sensor is operational and collecting data is shown by the red dotted line at the top.

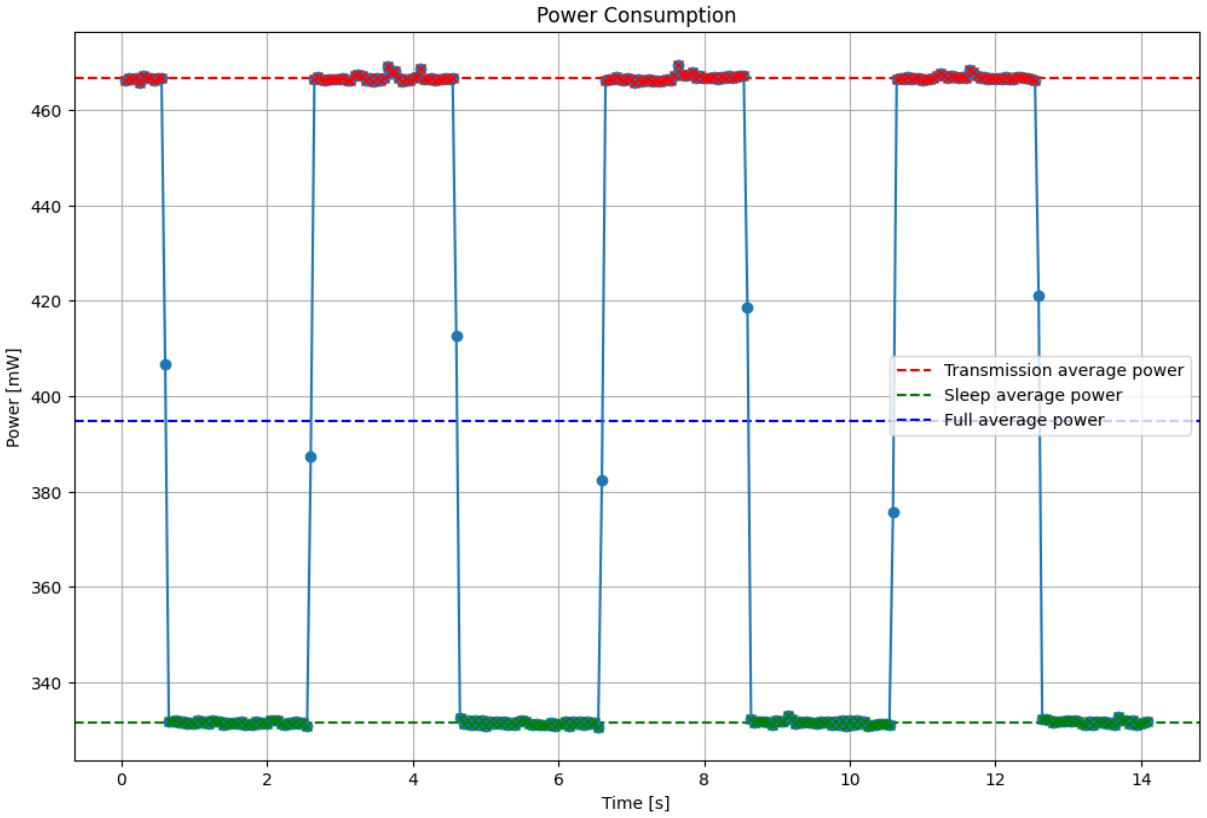

Fig. 4. Power Consumption (sensor_read.csv).

Table. 5

	Average Power Consumed [s]	Average Time Duration [W]
Sensor Reading	1.899114	0.886369
Idle	1.899208	0.629750

Table. 6

Duration of the cycle [s]	Energy used in cycle [J]	Number of cycles of battery	Battery lifetime [hours]
1.899114	1.5161198	13165	13.890

1. Comments Results and Improvements

The energy usage and battery life of the present parking sensor system are major problems. For a large-scale parking monitoring system, the insufficiently small battery size necessitates frequent changes, which is not feasible.

Optimizing the deep sleep cycle would be a major gain; a considerable increase might increase battery life from a few hours to more than a day, minimizing needless energy consumption when the parking place status stays the same. By lowering the frequency of high-power transmission states, batch processing and data compression could reduce the amount of data that is transmitted needlessly and save electricity.

Optimizing the system's transition into deep sleep mode is also crucial since it allows the system to enter low-power states more quickly by adjusting execution durations and delays. By putting these improvements into practice, the system can extend its battery life significantly, which makes it more appropriate for long-term deployment, especially in areas that are difficult to maintain or distant.