Chiara Thien Thao Nguyen Ba 10727985

Challenge

Code Explanation

The provided code uses the ESP32 microcontroller to interface with an ultrasonic sensor and ESP-NOW wireless communication protocol. The system aims to measure distance using the ultrasonic sensor and send a message indicating whether a parking slot is “OCCUPIED” or “FREE” based on a predefined threshold. The ESP32 goes into deep sleep after sending the message to conserve power.

At the beginning of the program, two pins are defined to handle the ultrasonic sensor: PIN_TRIG for sending pulses and PIN_ECHO for receiving the reflected signal. A constant, thDistance, is set to 50 cm as the threshold to determine whether the area being measured is “OCCUPIED” or “FREE”.
The TIME_TO_SLEEP constant is calculated through the given formula as: 85%50+5 = 40 seconds.
The ESP-NOW communication protocol is used to send the measurement data wirelessly to a sink, whose MAC address is specified in the broadcastAddress variable.

```
// Define the pins for the ultrasonic sensor
#define PIN_TRIG 13
#define PIN_ECHO 12
#define TIME_TO_SLEEP 40 // Time of deep sleep in seconds
#define uS_TO_S_FACTOR 1000000

uint8_t broadcastAddress[] = {0x8C, 0xAA, 0xB5, 0x84, 0xFB, 0x90}; // MAC address of the receiver
const int thDistance = 50; // Threshold distance in cm

esp_now_peer_info_t peerInfo;
```

The setup() function initializes the necessary components. The ultrasonic sensor is initialized using the initUltrasonicSensor() function, which configures the trigger pin as an output and the echo pin as an input.

```
// Function to initialize the ultrasonic sensor
void initUltrasonicSensor() {
  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECHO, INPUT);
}
```

```
void setup() {
  unsigned long startMillis, endMillis; // Start and end times

  Serial.begin(115200);

  // Initialize ultrasonic sensor
  Serial.println("Initializing ultrasonic sensor...");
  startMillis = millis();
  initUltrasonicSensor();
  endMillis = millis();
  Serial.println("Ultrasonic sensor initialized. Time elapsed: " + String(endMillis - startMillis) + " ms");
```

Next, the code retrieves the distance measured by the ultrasonic sensor through the getDistance() function. This function sends a pulse via PIN_TRIG, then listens for the returning echo on PIN_ECHO to calculate the time taken for the sound wave to return. Using this time, the distance to the object is calculated in centimeters.

```
// Function to get the distance using the ultrasonic sensor
float getDistance() {
  digitalWrite(PIN_TRIG, LOW);
  delayMicroseconds(5);
  digitalWrite(PIN_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG, LOW);

  int duration = pulseIn(PIN_ECHO, HIGH);
  float distance = duration / 58.0;
  Serial.print("Distance in cm: ");
  Serial.println(distance);
  return distance;
}
```

If the measured distance is less than or equal to the threshold (thDistance), the system determines the area is “OCCUPIED”. If the distance is greater than the threshold, the area is classified as “FREE”.
Once the distance is determined, a message is prepared. The message is either “OCCUPIED” or “FREE” depending on the comparison with the threshold.

```
// Get distance from the sensor
Serial.println("Getting distance from sensor...");
startMillis = millis();
float distance = getDistance();
endMillis = millis();
Serial.println("Distance measured. Time elapsed: " + String(endMillis - startMillis) + " ms");

// Prepare the message
String msg;
if (distance <= thDistance) {
  msg = "OCCUPIED";
} else {
  msg = "FREE";
}
```

Following this, the ESP-NOW protocol is initialized, setting the ESP32 to Station mode (WIFI_STA) to enable communication.

```
// Initialize ESP-NOW
Serial.println("Initializing ESP-NOW...");
unsigned long startWiFiMillis = millis();
WiFi.mode(WIFI_STA);
esp_now_init();
```

The receiver of the message is registered using its MAC address, stored in the peerInfo structure. The peer is added to ESP-NOW, allowing the ESP32 to send messages to this specific receiver. Two callback functions are registered: OnDataSent() and OnDataRecv(). OnDataSent() is used to verify whether the data transmission was successful, printing either “Ok” or “Error” based on the result. While the OnDataRecv() function is included for receiving data.

```
// Register peer
Serial.println("Registering ESP-NOW peer...");
startMillis = millis();
memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.channel = 0;
peerInfo.encrypt = false;
esp_now_add_peer(&peerInfo);
endMillis = millis();
Serial.println("Peer registered. Time elapsed: " + String(endMillis - startMillis) + " ms");

// Register callbacks
esp_now_register_send_cb(OnDataSent);
esp_now_register_recv_cb(OnDataRecv);
```

The ESP32 then sends the previous message to the registered peer device using esp_now_send(). After the message is sent, the device prepares to enter deep sleep mode.

```
// Send the message
Serial.println("Sending ESP-NOW message: " + msg);
startMillis = millis();
esp_now_send(broadcastAddress, (uint8_t*)msg.c_str(), msg.length() + 1);
endMillis = millis();
Serial.println("Message sent. Time elapsed: " + String(endMillis - startMillis) + " ms");
```

To conserve power, the ESP32 enters deep sleep for 40 seconds using the function esp_sleep_enable_timer_wakeup(). Before going to sleep, the WiFi module is turned off to minimize power consumption. The microcontroller will wake up after 40 seconds, reinitialize, and repeat the process of measuring the distance and sending a new message.

```
// Enable deep sleep
Serial.println("Preparing for deep sleep...");
unsigned long endWiFiMillis = millis();
Serial.println("WiFi time on: " + String(endWiFiMillis - startWiFiMillis) + " ms");
WiFi.mode(WIFI_OFF);
delay(1000);
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
Serial.println("Going to sleep now.");
Serial.flush();
esp_deep_sleep_start();
```

Power and Energy Consumption Estimation

By running Wowki simulation and looking at the prints of time elapsed for each stage, I obtained the following time estimations for each stage:

$$T_{IDLE} = 1\,ms$$
$$T_{READ} = 5\,ms$$
$$T_{WIFI-ON} = 200\,ms$$
$$T_{TX} = 1\,ms$$
$$T_{WIFI-OFF} = 1\,s$$
$$T_{DEEPSLEEP} = 40\,s$$

By looking at the csv files of timestamps and power consumptions, I retrieved the following average power consumptions for each stage:

$$P_{IDLE} = 313\,mW$$
$$P_{READ} = 466\,mW$$
$$P_{WIFI-ON} = 777\,mW$$
$$P_{TX} = 1210\,mW$$
$$P_{WIFI-OFF} = 307\,mW$$
$$P_{DEEPSLEEP} = 59\,mW$$

For each stage, I calculated the energy consumption as $E = T \cdot P$ and I obtained the following results:

$$E_{IDLE} = 3,13 \cdot 10^{-4} J$$
$$E_{READ} = 2,33 \cdot 10^{-3} J$$
$$E_{WIFI-ON} = 0,155\,J$$
$$E_{TX} = 1,21 \cdot 10^{-3} J$$
$$E_{WIFI-OFF} = 0,307\,J$$
$$E_{DEEPSLEEP} = 2,36\,J$$

Thus, the estimated energy consumption of one transmission cycle is:

$$E_{cycle} = E_{IDLE} + E_{READ} + E_{WIFI-ON} + E_{TX} + E_{WIFI-OFF} + E_{DEEPSLEEP} = 2,8\,J$$

The total time of one transmission cycle is:

$$T_{cycle} = T_{IDLE} + T_{READ} + T_{WIFI-ON} + T_{TX} + T_{WIFI-OFF} + T_{DEEPSLEEP} = 41,2\,s$$

The energy of the battery is calculated given the requirements as:

$$E_{battery} = 7985\%5000 + 15000 = 17985\,J$$

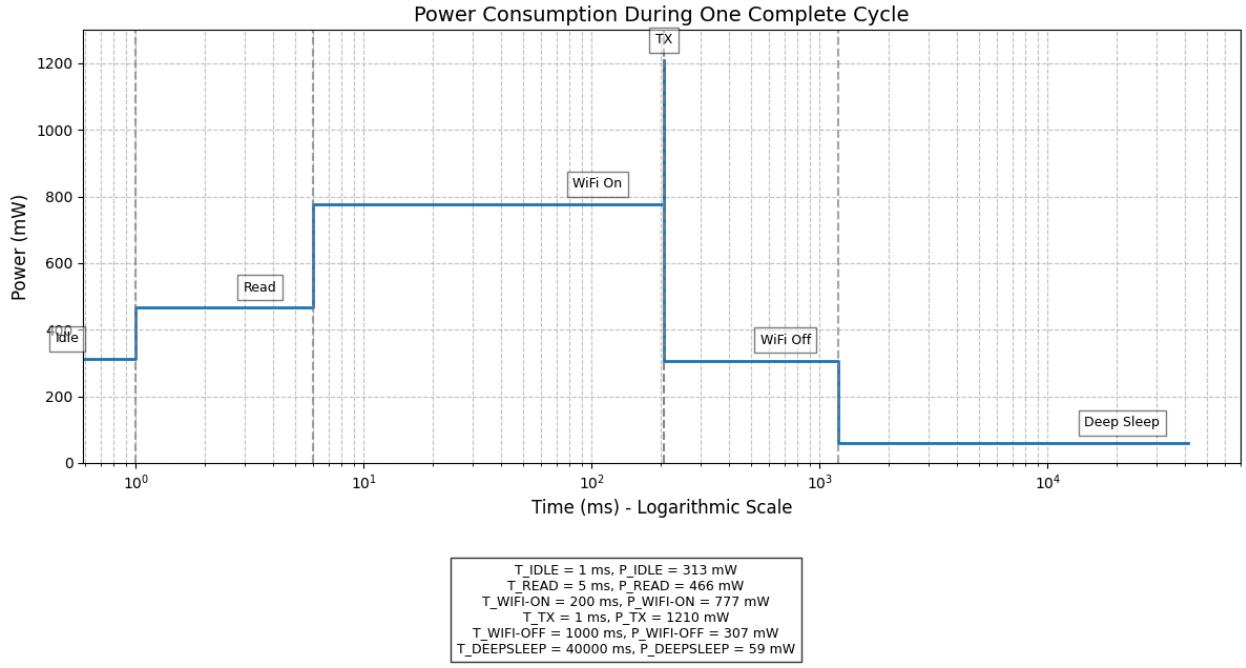
Given the energy consumption of one transmission cycle and the energy of the battery, I calculated the total number of cycles till the battery lasts as:

$$N = \frac{E_{battery}}{E_{cycle}} \simeq 6423\,cycles$$

In conclusion, the estimated time the sensor node lasts before changing the battery is:

$$T = N \cdot T_{cycle} \simeq 264628\,s \simeq 3\,days$$

The following picture illustrates the graph of power-time of one complete transmission cycle showing for each stage the time and the power consumed.



Possible Improvements of Sensor Node

To improve the energy efficiency of the parking sensor node, some main optimizations have been considered:

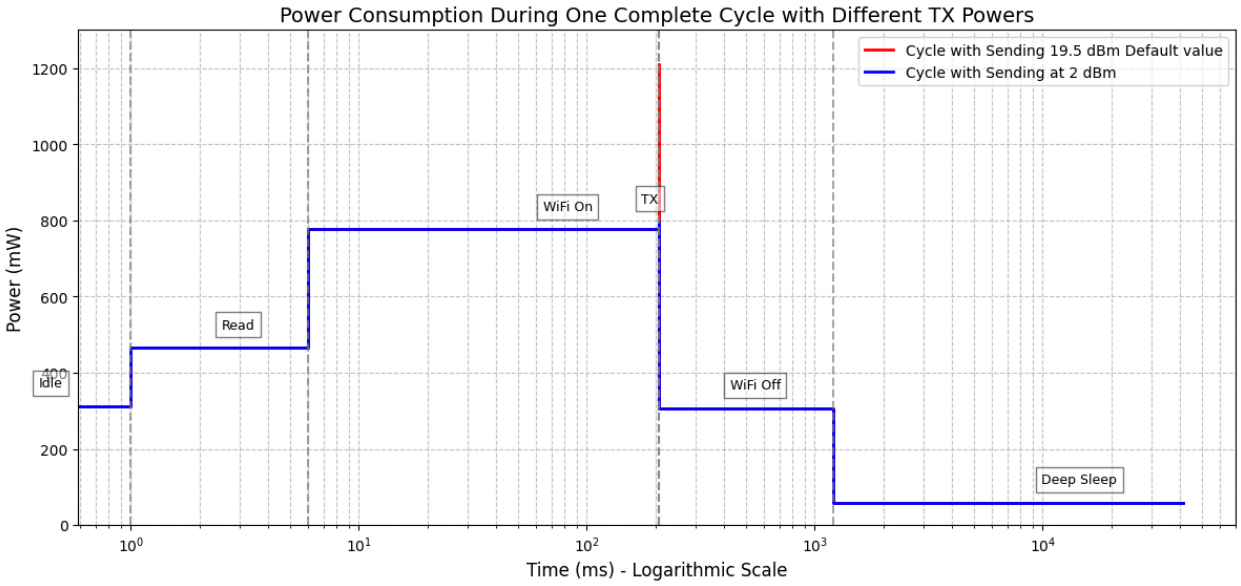
- By lowering the WiFi transmission power from the default value of 19.5 dBm to 2 dBm, we can significantly reduce the energy consumed during data transmission. The transmission power is directly proportional to the energy required for communication, and reducing this power still ensures reliable communication within the necessary range for the parking sensor application. This adjustment lowers the overall energy consumption without compromising functionality.
- Instead of transmitting data to the sink node at regular intervals or after every occupancy check, the system has been modified to activate the WiFi and send data only when there is a change in the parking spot's state (from "FREE" to "OCCUPIED" or vice versa). If no change is detected, the WiFi remains off, and no message is sent. This approach ensures that the sensor only consumes energy for WiFi activation and data transmission when absolutely necessary, reducing redundant transmissions and significantly saving energy during periods of no state change. The sink node assumes that if no update is received, the parking state has remained the same.
- By increasing the deep sleep time period, we can reduce the frequency of wake-ups and have a higher estimated time for the duration of the sensor node. For instance, instead of waking up every 40 seconds, waking up every few minutes can drastically cut down on energy usage.

Based on the first above consideration (i.e. by considering the WiFi transmission power be 2 dBm), the estimated power and energy consumption of the transmission would be:

$$P_{TX} = 797\ mW$$

$$E_{TX} = 7,97 \cdot 10^{-4} J$$

The following picture illustrates the change in the transmission power from the default value at 19.5 dBm (as in the previous case) to a lower value of 2 dBm.



By considering the case when the state of the parking slot does not change, so the WiFi remains off and there is no transmission of the message to the sink, we have that the total estimated energy consumption of one transmission cycle would be:

$$E_{cycle} = E_{IDLE} + E_{READ} + E_{DEEPSLEEP} = 2,36\,J$$

Thus, in this case we would have for a single cycle a saving of 15,71% of energy consumption compared to the initial case.

The following graph illustrates one cycle of power consumption when the WiFi is not activated and no message is sent to the sink since the park slot did not change its status from the previous wakeup.