Internet of Things – First Challenge

Abate Kevin Pio: 10812892

Pigato Lorenzo: 10766953 [Team Leader]

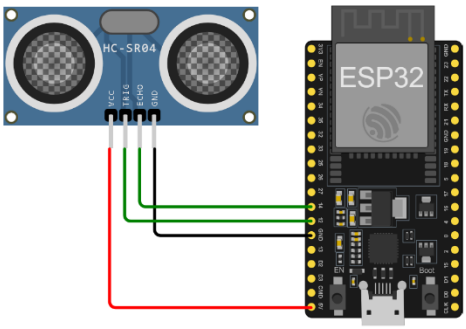
Project Topic

The project consists of developing a simple **parking occupancy node** using an **ESP32 board** paired with the **HC-SR04** ultrasonic distance sensor using the ESP-NOW protocol for communicating between nodes and the central sink.

Schematics

The schematics used to connect the ESP32 board with the HC-SR04 are reported below:

HC-SR04	ESP32
VCC	+5V
TRIG	Pin 12
ECHO	Pin 14
GND	GND



Code Review

Constants, global variables and function prototypes

```
#include <WiFi.h>
#include <esp_now.h>

/*
  Internet of Things - First Challenge
  Date: 11-03-2025 / 20-03-2025

  Abate Kevin:    10812892
  Pigato Lorenzo: 10766953 [Team Leader]

  Project Title: Parking occupancy node
*/

// Distance sensor HC-SR04 pinout

#define TRIG_PIN 12
#define ECHO_PIN 14

// Constants

#define SLEEP_TIME_SEC  8       // Formula: 53 % 50 + 5 = 8
#define S_uS_CONV 1000000       // Conversion factor from us to s

long long int startTime;

uint8_t broadcastAddress[] = {0x8C, 0xAA, 0xB5, 0x84, 0xFB, 0x90};

esp_now_peer_info_t ESP_sink; // peer data

// -- Functions' prototypes -- //
float getDistance();
void wifiInit();
void onDataRecv(const uint8_t *mac_addr, const uint8_t *data, int data_len);
void onDataSent(const uint8_t *mac_addr, esp_now_send_status_t status);
void sendMessage(String message);
// ------------------------- //

```

In this first part of code, all constants and global variables are defined

To keep the code clean, function prototypes have been defined before actual function implementation

setup()

```
38  void setup()
39  {
40      startTime = micros();
41      // Open serial port: 115200
42      Serial.begin(115200);
43
44      Serial.printf("[0] -- Booting\n");
45
46
47      // HC-SR04 configuration
48      pinMode(TRIG_PIN, OUTPUT);
49      pinMode(ECHO_PIN, INPUT);
50
51      Serial.printf("[%d] -- Setup completed\n", micros() - startTime);
52
53  }
54
```

Setup function contains all the instructions that must be run once at every boot of the board.

Every time the board starts, *startTime* variable is initiated with the current board uptime in microseconds, which is worth mentioning that does not get reset after the board enters deep sleep.

Serial communication and board pins are also initialized in this step.

loop()

```
55  void loop()
56  {
57      // -- loop will be executed only once per cycle due to deep sleep call -- //
58
59      // Fetch distance
60      float distance = getDistance();
61
62      // Check parking status
63      String message = distance < 50 ? "OCCUPIED" : "FREE";
64      Serial.printf("[%d] -- Sensor reading complete\n", micros() - startTime);
65
66      // Configure WiFi for transmission and send message to sink
67      wifiInit();
68      Serial.printf("[%d] -- Wifi initialized\n", micros() - startTime);
69
70      sendMessage(message);
71      Serial.printf("[%d] -- Message sent\n", micros() - startTime);
72
73      delay(50);
74
75      WiFi.mode(WIFI_OFF);
76      Serial.printf("[%d] -- WiFi turned off\n", micros() - startTime);
77
78      Serial.printf("[%d] -- Entering deep sleep...\n", micros() - startTime);
79      Serial.flush(); // Flush serial buffer before entering deep sleep
80
81      delay(50);
82
83      // Set wakeup timer
84      esp_sleep_enable_timer_wakeup(SLEEP_TIME_SEC * S_uS_CONV);
85      esp_deep_sleep_start();
86  }
87
```

Loop function is usually executed more than once, but due to the board entering deep sleep at the end of every cycle, in this case it's executed just once per power cycle. This means that the code which is run here could be placed inside the *setup()* function instead, but keeping it in the *loop()* makes the code conceptually easier to understand.

During the execution of the *loop()*, the board reads from the sensor using the *getDistance()* function, then initializes the WiFi module, sends the message with the parking spot status to the sink node and finally turns off Wifi module before entering deep sleep.

getDistance()

```
91  float getDistance()
92  {
93
94      digitalWrite(TRIG_PIN, HIGH); // Start emitting signal
95      delayMicroseconds(10);
96      digitalWrite(TRIG_PIN, LOW); // Stop emitting signal
97
98      // Calculate the time occurred between sending and receiving back the signal
99      int duration = pulseIn(ECHO_PIN, HIGH);
100
101     // Convert the result in cm according to sensor's datasheet
102     float distance = duration / 58.3;
103
104     Serial.printf("Distance: %.2f\n", distance);
105
106     return distance;
107 }
```

This function is used to get a distance measure using the HC-SR04 ultrasonic sensor.

The *trigger* pin is set to high for 10 microseconds to emit the ultrasonic wave, then *pulseIn()* returns the time taken for the *echo* pin to toggle from low to high. This is the time that it takes for the sound wave to cover the distance from the sensor to the object and back. According to the datasheets found online, to get a distance in millimeters it's necessary to divide by 58.3

Abate Kevin Pio: 10812892

Pigato Lorenzo: 10766953

wifiInit()

```
109  void wifiInit()
110  {
111      Serial.printf("Initializing WiFi...\n");
112
113      WiFi.mode(WIFI_STA);
114      WiFi.setTxPower(WIFI_POWER_2dBm); // Set wifi transmission power
115      esp_now_init();                   // Init ESP-NOW communication protocol
116
117      esp_now_register_recv_cb(onDataRecv); // Set receiving callback
118      esp_now_register_send_cb(onDataSent); // Set sending callback
119
120      // Peer registration
121      memcpy(ESP_sink.peer_addr, broadcastAddress, 6);
122      ESP_sink.channel = 0;
123      ESP_sink.encrypt = false;
124
125      esp_now_add_peer(&ESP_sink); // Add sink to peers
126  }
```

This snippet illustrates how the WiFi circuitry is turned on and the communication protocol is set up.

At first, the WiFi module is switched on and put in Station mode, then transmission power is set to 2dBm to waste as little energy as possible, assuming the parking lot to be small enough to have the sink node nearby, reachable by such a low power signal.

Callback functions are then set for sending and receiving messages; however this is for simulation purposes only because nodes shouldn't get any message by other nodes or from the sink.

Finally, the sink is registered as a peer in ESP-NOW communication protocol, using the broadcast MAC address, channel 0 and not encrypted connection.

Callback functions

```
128  // Receiving callback (for simulation purpouse)
129  void onDataRecv(const uint8_t *mac_addr, const uint8_t *data, int data_len)
130  {
131      char rcvstring[data_len];
132      memcpy(rcvstring, data, data_len);
133
134      Serial.printf("Message received: %s\n", String(rcvstring));
135  }
136
137  // Sending callback
138  void onDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)
139  {
140      Serial.printf("Message status: ");
141      Serial.printf(status == ESP_NOW_SEND_SUCCESS ? "SENT\n" : "ERROR\n");
142  }
143
144  // Send message through WiFi with ESP-NOW protocol
145  void sendMessage(String message)
146  {
147
148      uint8_t *addr = broadcastAddress;
149      uint8_t *data = (uint8_t *)message.c_str(); // Casting string to uint8_t
150
151      int len = message.length() + 1;
152
153      esp_now_send(addr, data, len);
154  }
```

These functions take care of handling message dispatch.

When a message is sent through *esp_now_send()*, *onDataSent()* is called and the status of the sending is reported on the serial output, while on message receive *onDataReceive()* is called and the content of the message received is copied onto the buffer and printed on serial output.

Online simulation

The code is also available online at: https://wokwi.com/projects/425605103919355905, with some minor differences with the code that can be found alongside this document due to the different versions of ESP-NOW protocol libraries in the online simulator with the ones in the provided virtual machine.

Abate Kevin Pio: 10812892

Pigato Lorenzo: 10766953

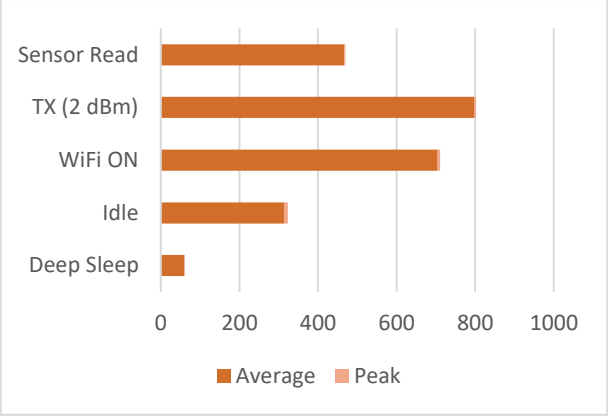
Energy Consumption and Average Power Estimation

In order to compute energy consumption and the average power estimation, a Jupyter Notebook has been used and can be found in the folder alongside the report.

Average Power Consumption

The analysis of average power consumption is based upon the CSV files provided which contain the data regarding each operational state of the ESP32 board. Data have been filtered using threshold values which were empirically estimated plotting the given dataset. The results are as follows:

State	Dataset	Peak	Average
Deep Sleep	deep_sleep.csv	61.12	59.66 mW
Idle (WiFi OFF)		333.04	313.40 mW
Idle (WiFi ON)	send_different_TX.csv	711.82	704.22 mW
Transmitting (2 dBm)		802.91	797.29 mW
Sensor Read (WiFi OFF)	read_sensor.csv	469.49	466.74 mW



Energy Consumption

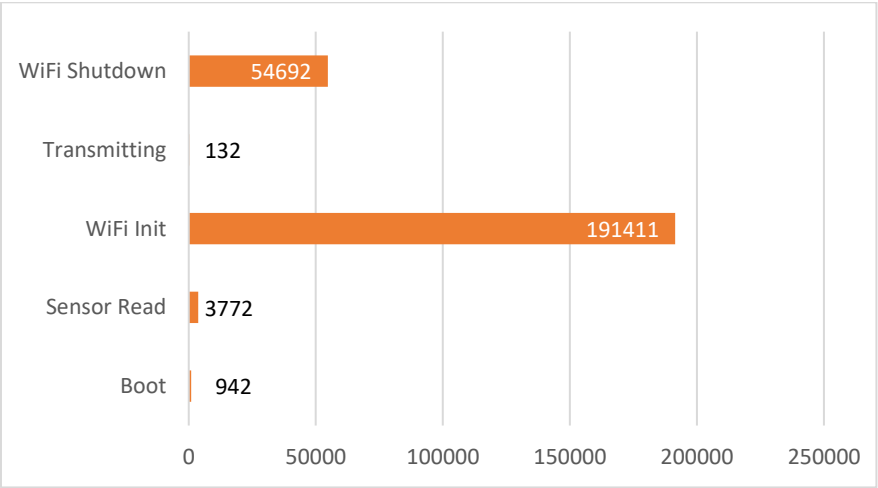
Power consumption data was used alongside the time estimates derived by the simulation to calculate the average energy consumption for a complete duty cycle.

Time Estimation

Data related to time of working in each operational state has been obtained exploiting the *micros()* function, which returns the number of microseconds elapsed from the power up of the system. As said before, *micros()* does not reset when deep sleep is entered, so at every awakening a new baseline time was collected and then subtracted before printing out completion time of each operational state.

Time estimates are not highly accurate due to the fact that simulations aren't reliable enough to give consistent time results, so data outliers were filtered and interpolated to reduce variance. Data can be found in *timing.txt*, results are reported below:

Operation	Average Time
Boot and Setup (Idle – WiFi OFF)	942 µs
Sensor read	3772 µs
WiFi initialization (Idle - WiFi ON)	191411 µs
Transmitting	132 µs
WiFi Shutdown (Idle - WiFi ON)	54692 µs
Deep Sleep	8 s

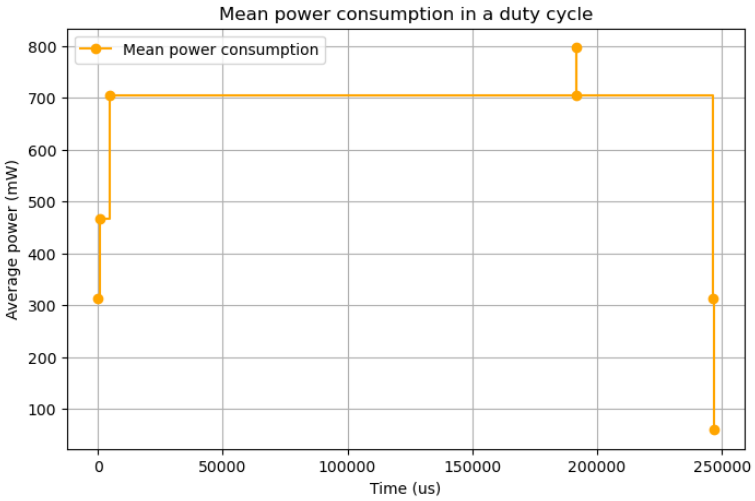


Abate Kevin Pio: 10812892

Pigato Lorenzo: 10766953

Energy estimation

Given the time estimations and power consumptions obtained before, it's possible to plot the average power as a function of time. The integral of such a function is the energy consumption of the ESP32 module. This integral is easily computable as the product of average power of the operational state and the duration of it:



Operation	Average Time
Deep Sleep	477.28 mJ
Idle – WiFi OFF	0.295 mJ
Sensor Reading	1.761 mJ
Idle – WiFi ON	173.265 mJ
Transmission	0.105 mJ
Total Energy Consumption	652.7 mJ

Battery Life Estimation

Based on the estimated energy consumption, a battery of 16953 Joule will last 25973 cycles or, equivalently, 59.53 hours, according to the formula:

$$cycles = \frac{battery}{total\ energy\ consumption} = \frac{16953\ [J]}{652.7\ [mJ]} = 25973$$

$$uptime = cycles\ \cdot cycleTime = \frac{25973\ \cdot 8.250[\mathrm{s}]}{3600} = 59.53\ h$$

Possible Improvements

Transmitting only at status change

To reduce energy consumption, transmission of updates towards the sink should occur only when the parking status changes. Initializing the WiFi module to communicate and idling with WiFi turned on are the most energy consuming task, so they must be avoided when not strictly necessary.

This modification would dramatically improve the lifetime of the single node because no energy would be wasted in turning on and off the WiFi module and transmitting redundant data. The improvement is easily achievable with little effort adding a *prevMessage* variable, which will contain the previous message sent to the sink. If *prevMessage* matches the current *message* value, there is no need to start communicating with the sink.

Simulating power consumption of this alternative would be of no use, a real scenario would be needed to appreciate the differences. Code for this version is provided in the *improved.ino* file, which can be found in the *src* folder.

Abate Kevin Pio: 10812892

Pigato Lorenzo: 10766953

Longer sleep time

Since a parked car remains in place for long periods of time, deep sleep duration can be increased when the sensor detects that the parking spot has been occupied. Polling every 8 seconds is too frequent for this kind of application, a frequency of a minute should be good enough to fit the requirements. Power consumption would be as follows:

$$cycles = \frac{battery}{total\ energy\ consumption} = \frac{16953\ [J]}{175.43\ [mJ] + (59.66[mW]\ \cdot 60[s])} = 4514.7$$

$$uptime = cycles\ \cdot cycleTime = \frac{4514.7\ \cdot\ 60.250[\mathrm{s}]}{3600} = 75.55\ h$$

Alternative technologies and protocols

Other technologies and protocols could offer better energy efficiency than the ESP-NOW protocol or WiFi technology. Different, more efficient, sensors can also be considered.

Abate Kevin Pio: 10812892

Pigato Lorenzo: 10766953