POLITECNICO DI MILANO
Master of Science in Automation and Control Engineering

Report of

Challenge 01

Antonello Lagalante

Person code: 10938409

Wokwi project (click to open)

Contents

Chapter 1

Explanation of the code logic

1.1 Hardware scheme

Only two objects had to be used for the simulation of the system: the **ESP32** and the **HC-SR04 Ultrasonic Distance Sensor**. They are connected to each other as depicted in the figure below:

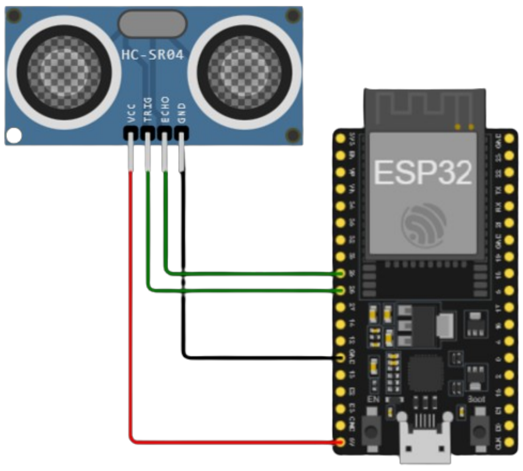

Figure 1.1: Scheme of the hardware used in the simulation.

1.2 Libraries and parameters definition

The code we refer to in this section is the following one:

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

// Define the pin connections for the HC-SR04 sensor
#define PIN_TRIG 26
#define PIN_ECHO 25

#define uS_TO_S_FACTOR 1000000
#define TIME_TO_SLEEP  14           //Duty cycle (changes according to the
    PersonCode)

#define PARKING_SLOT_IDENTIFIER 1 // Each parking slot has its own
    identifier
```

```

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

esp_now_peer_info_t peerInfo;
```

The first thing to do is to add the two standard libraries needed for the WiFi of the ESP32 and the ESP-NOW protocol for the communication between two ESP32s.

Then, since the TRIG and ECHO pin of the HC-SR04 are respectively connected to the pin 26 and 25, we can define them with two variables.

The time during which the microcontroller must be in deep-sleep is also defined through a variable and is equal to 14 seconds as calculated by the following formula:

$$\boldsymbol{X} = 09\%50 + 5 = 14$$

Since with this project we simulate a single parking occupancy node, we can also define for each node a numeric identifier from 1 to $\boldsymbol{N}$, where $\boldsymbol{N}$ is the number of parking spaces. In this case we are considering the parking slot number 1.

Finally, we define the address of the sink node to which the status of the parking slot is to be sent. In this case, to make the ESP-NOW protocol work with the Wokwi emulator, a Broadcast address (8C:AA:B5:84:FB:90) is used and the transmission and reception of messages happens on the same board.

1.3 Callback functions

The code we refer to in this section is the following one:

```
// Callback function to handle data send event
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status){
  Serial.print("Send status: ");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Ok" : "Error");
}

// Callback function to handle data received event
void OnDataRecv(const uint8_t *mac, const uint8_t *data, int len){
  char receivedString[len];
  memcpy(receivedString, data, len);

  String parkIdent = String(receivedString[0]);
  String parkStatus = "";
  for (int i = 1; i < len; i++){
    parkStatus = parkStatus + String(receivedString[i]);
  }

  Serial.println("Parking slot " + parkIdent + " is " + parkStatus);
}
```

Sending the message with the parking slot identifier and status to the sink node is an event that triggers the execution of the *OnDataSent* function, which prints 'Ok' if the sending was successful or 'Error' if not.

In this case, since we are simulating the sink node with the same ESP32 that sends the message, the receipt of one message will trigger the execution of the *OnDataRecv* function which will print out a message with the identifier of the parking slot from which the message was sent (first digit) and its status (FREE/OCCUPIED).

1.4 Distance reading and state definition

The code we refer to in this section is the following one:

```
// Function that reads the sensor data
float readDistance() {
  digitalWrite(PIN_TRIG, LOW);
  delayMicroseconds(2);
  digitalWrite(PIN_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG, LOW);
  int duration = pulseIn(PIN_ECHO, HIGH);
  return duration * 0.034 / 2; // Conversion in cm
}

// Function that define the parking slot status
String defineStatus(float distance){
  String status = "FREE";
  if (distance < 50) status = "OCCUPIED";

  return status;
}
```

The HC-SR04 works in a simple way: we start a new distance measurement setting the TRIG pin to high for 10uS or more. Then wait until the ECHO pin goes high, and count the time it stays high (pulse length). The length of the ECHO high pulse is proportional to the distance [1]. The function *readDistance* implements this logic, converting the distance obtained into centimetres.

The function *defineStatus* returns the status of the parking slot (FREE/OCCUPIED) according to the distance read by the sensor (the value returned by the previous function).

1.5 Sending the message and enabling deep-sleep mode

The code we refer to in this section is the following one:

```
void setup() {
  Serial.begin(115200);
  delay(100);

  // Pin modes for the HC-SR04 sensor
  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECHO, INPUT);

  // Performing the measurement and determining the parking slot status
10   String message = (String)PARKING_SLOT_IDENTIFIER + defineStatus(
       readDistance());
11
12   // Turning on WiFi and sending the parking status with esp-now
13   WiFi.mode(WIFI_STA);
14   esp_now_init(); // Initiate the ESP-NOW protocol on the board
15
16   esp_now_register_send_cb(OnDataSent);
17   esp_now_register_recv_cb(OnDataRecv);
18
19   memcpy(peerInfo.peer_addr, broadcastAddress, 6);
20   peerInfo.channel = 0;
21   peerInfo.encrypt = false;
22   esp_now_add_peer(&peerInfo);
23
24   esp_now_send(broadcastAddress, (uint8_t*)message.c_str(), message.
       length()+1);
25   delay(100);
26
27   // Disabling Wi-Fi and going to deep sleep
28   WiFi.mode(WIFI_OFF);
29   delay(100);
30   Serial.println("Going to sleep now for " + String(TIME_TO_SLEEP) + "
       seconds.");
31
32   //Since Wokwi does not simulate the sleep time, we can use a delay to
       check if the code works properly.
33   //delayMicroseconds(TIME_TO_SLEEP*uS_TO_S_FACTOR);
34
35   esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
36   Serial.flush();
37   esp_deep_sleep_start();
38 }
39
40
41 void loop() {
42  //This is not going to be called
43 }
```

After defining the pin modes for the HC-SR04 sensor, we create the string to be sent to the sink node using the two functions defined in the previous section.

We turn on the Wifi and we initialise the ESP-NOW protocol on the board. At this point we can attach the function *OnDataSent* and *OnDataRecv* defined above to the ESP-NOW module to "tell" the board what to do when a message is sent and received.

With the code from line 19 to 24 we configure the node to which the message is to be sent, indicating the broadcast address, the channel and whether to encrypt the message. At this point we can send the message to the sink node.

Finally, we switch off the Wifi module and send the board into deep sleep for 14 seconds, after which the board will wake up to repeat the cycle.

Chapter 2

Energy Consumption Estimation

2.1 Power consumption of each state

By printing the milliseconds in which Wokwi performs each command, it is possible to have an estimate of how long each state lasts (Deep Sleep state, Idle, Transmission state, Sensor reading).

Using this data and the information in the .csv files provided, it is possible to schematise the power output trend (transmitting at 19.5 dBm) as follows:

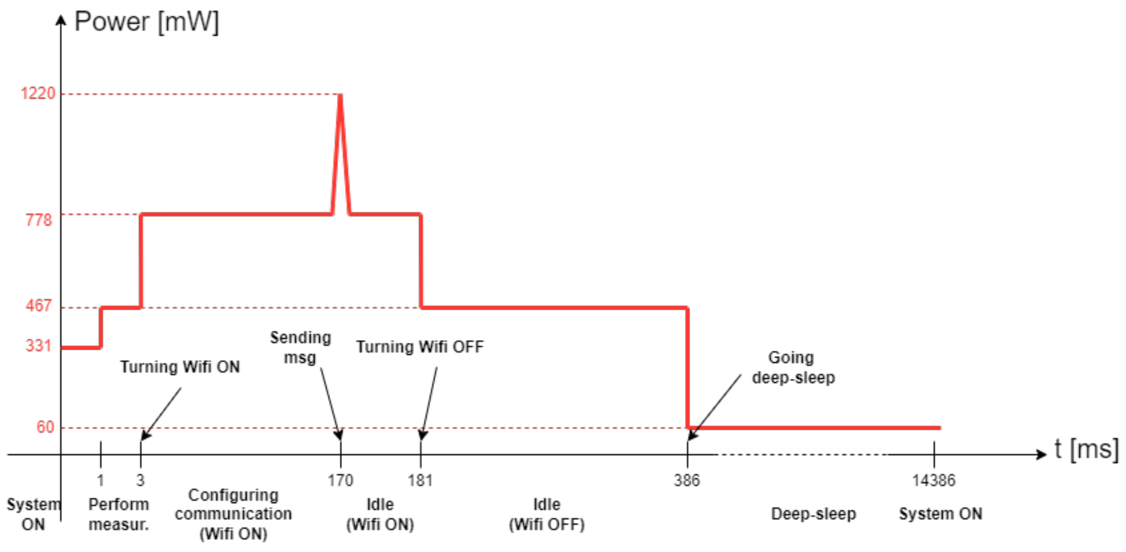

Figure 2.1: Power trend over time.

From this scheme we have:

- Idle (Wifi OFF): 331 mW
- Sensor reading: 467 mW
- Configuring communication (Wifi ON): 778 mW
- Transmission: 1220 mW
- Idle (Wifi ON): 778 mW
- Idle (Wifi OFF): 467 mW
- Deep-sleep: 60 mW

2.2 Energy consumption of 1 transmission cycle

Using the data determined in the previous section, multiplying the average power by the duration of each state, it is possible to determine the power consumption for 1 transmission cycle:

$$\begin{gathered}\boldsymbol{E} = 0.331 \cdot 0.001 + 0.467 \cdot 0.002 + 0.778 \cdot 0.167 + 1.22 \cdot 0.001 + \\ + 0.778 \cdot 0.01 + 0.467 \cdot 0.205 + 0.06 \cdot 14 = 1.0759\ \boldsymbol{J} = 1075.9\ \boldsymbol{mJ}\end{gathered}$$

2.3 Sensor node lifetime

Using a battery that contains $\boldsymbol{Y} = 8409 + 5 = 8414$ Joule of energy, the sensor node can do $8414/1.0759 \approx 7820$ cycles. Since one cycle takes 14.386 seconds to be completed, the node can operate for 112500 seconds (≈ 31.25 hours) before changing the battery.

Chapter 3

Results and Improvements

3.1 Results

With the implemented system, it is necessary to change the battery of each sensor every approximately 32 hours, which may not be a simple procedure in the case of a car park with hundreds of parking spaces and open 24 hours a day.

3.2 Improvements

Q: Starting from the system requirements, propose some possible improvements in terms of Energy Consumption without modifying the main task «Notify to a Sink node the occupancy state of a parking spot».

Possible improvements to the system could be:

1. If the maximum distance of the parking slots from the sink node is rather low, we can transmit the messages with a lower power up to 2 $\boldsymbol{dBm}$. In this case, in order to calculate the minimum power at which we can transmit, we can either base it on the car park with the maximum distance from the sink node (in case we want to change the batteries of all the sensors at the same time) or use a lower power for the closer sensors and a higher power for the more distant ones.
2. We can use sensors that consume less energy in the distance acquisition with respect to the HC-SR04.
3. The distance acquisition frequency of 14 seconds is rather high if we consider the application context of determining whether a parking space is free or not. A more appropriate frequency might be to take the measurement every minute, which would send the ESP32 into deep-sleep for much longer time, consuming much less energy.