POLITECNICO
MILANO 1863

Simple parking occupancy node using Wokwi

1^{st} IoT challenge

Author
Christian Confalonieri

Professors
Matteo Cesana
Antonio Boiano

The Wokwi project can be found on: **https://wokwi.com/projects/391892882305438721**.

2023/2024

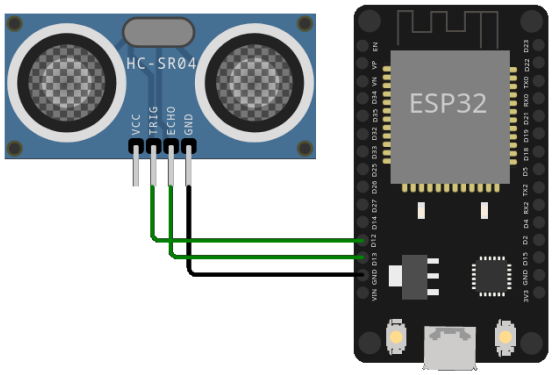
Contents

1 Introduction

This document describes the design and implementation of a simple parking occupancy node using **Wokwi**, a web-based platform that allows you to simulate and test various IoT and embedded system projects using ESP32, STM32, Arduino, Raspberry Pi Pico and other devices.
The node is designed to detect the presence of a car in a parking spot and send the information to a central ESP32 sink node (star topology), minimizing the power consumption and the energy used.
To achieve this, the node is equipped with a **HC-SR04** ultrasonic distance sensor and the **ESP-NOW** protocol is used to send the information to the sink node.
In order to minimize the power consumption, the node is designed to be in **deep sleep** mode most of the time, waking up periodically, with a defined duty cycle, to check the presence of a car.

2 Design and Implementation using Wokwi

As mentioned before, the node is equipped with an **HC-SR04** ultrasonic distance sensor and an **ESP32** microcontroller:



2.1 Description of the components

- **HC-SR04** ultrasonic distance sensor: it is a sensor that uses ultrasonic waves (sonar) to measure the distance between the sensor and an object. It consists of two ultrasonic transducers: a transmitter and a receiver. The transmitter (**trig pin**) transmits a burst of ultrasonic waves, which are reflected by an object (in our case, the car) and then received by the receiver (**echo pin**). The time it takes for the waves to travel back to the receiver is used to calculate the distance between the sensor and the object:
  - Centimeters: distance = PulseMicros/58
  - Inches: distance = PulseMicros/148

  This sensor reads from **2cm to 400cm** (0.8inch to 157inch) with an accuracy of 0.3cm (0.1inches), which is good for most hobbyist projects.
  More informartion about the implementation on Wokwi can be found on the official website.
- **ESP32** microcontroller: it is a low-cost, low-power system on a chip microcontroller with integrated Wi-Fi and dual-mode Bluetooth. It is designed to be used in various IoT applications.
  More informartion about the implementation on Wokwi can be found on the official website.

2.2 Explanation of the code logic

The Wokwi project can be found on: **https://wokwi.com/projects/391892882305438721**.

2.2.1 Libraries and global variables

The code starts by including the necessary libraries:

- **WiFi.h**: the Wi-Fi library is used to connect to the Wi-Fi network.
- **esp_now.h**: the ESP-NOW protocol is used to send the information to the sink node.

Then, the global constants and variables are defined:

- **ECHO_PIN** and **TRIG_PIN**: the pins used to connect the HC-SR04 to the ESP32 (in this case, respectively **13** and **12**).
- **MAX_DISTANCE**: the maximum distance where a parking spot is considered occupied (in this case, **<50** cm).
- **uS_TO_S_FACTOR**: the factor used to convert microseconds to seconds (in this case, $\mathbf{10^6}$).
- **TIME_TO_SLEEP**: the time the node is in deep sleep mode (in this case, **51%50+5=6**).
- **broadcastAddress[]**: the address of the sink node. Since in Wokwi we can't use the MAC address of another ESP32, we use instead the broadcast address: **{0x8C, 0xAA, 0xB5, 0x84, 0xFB, 0x90}**.
- **esp_now_peer_info_t peerInfo**: the structure used to store the information about the sink node.

2.2.2 Auxiliary functions

- **timestamp()**: it returns a string timestamp in the format [day:#days time:HH:MM:SS.SSS].

Communication functions:

- **onDataSent()**: it is called when the data has been sent by the ESP-NOW protocol. In the implementation provided it simply print the transmission result (OK or ERROR) and the timestamp.
- **onDataRecv()**: it is called when the data has been received by the ESP-NOW protocol. In the implementation provided it simply print the received data and the timestamp.
  Notice that in the Wokwi implementation the node is both the sender and the receiver of the data, so we need this function, but in a real implementation the sink node would be the receiver of the data.

Sensor functions:

- **readDistanceCM()**: it reads the distance from the HC-SR04 sensor and returns it in centimeters.
- **isOccupied()**: it calls readDistanceCM() and prints the distance. If the distance is less than MAX_DISTANCE, it returns "OCCUPIED", otherwise it returns "FREE".

2.2.3 Setup and loop functions

2.2.3.1 Setup function

The setup function due to the implementation of deep sleep mode is the main function of the code. It is called only once when the node is powered up or reset, and it contains all the logic of the node. The function starts by initializing the serial communication and the Wi-Fi connection. Then, it initializes the ESP-NOW protocol and sets the onDataSent() and onDataRecv() functions.
After that, it sets the peerInfo structure with the broadcast address and the ESP-NOW channel.
After that, it sets the pins of the HC-SR04 sensor, calls the isOccupied() and sends the result to the sink node, with a transmission power of 2dBm.
Finally, it power off the Wi-Fi and enters deep sleep mode for TIME_TO_SLEEP seconds.
More in detail, the function does the following (I will exclude all the prints for brevity):

- **Serial.begin(115200)**: it initializes the serial communication with a baud rate of 115200.
- **WiFi.mode(WIFI_STA)**: it sets the ESP32 as a station that connects to an access point.
- It sets the ESP-NOW protocol:
  - **esp_now_init()**: it initializes the ESP-NOW protocol.
  - **esp_now_register_send_cb(onDataSent)**: it sets the onDataSent() function.
  - **esp_now_register_recv_cb(onDataRecv)**: it sets the onDataRecv() function.
- It sets and adds the peer:
  - **memcpy(peerInfo.peer_addr, broadcastAddress, 6)**: it sets the peerInfo structure with the broadcast address.
  - **peerInfo.channel = 0**: it sets the ESP-NOW channel to 0.
  - **peerInfo.encrypt = false**: it sets the encryption to false.
  - **esp_now_add_peer(&peerInfo)**: it adds the peerInfo structure to the ESP-NOW protocol.
- It sets the pins of the HC-SR04 sensor and calls the isOccupied() function:
  - **pinMode(ECHO_PIN, INPUT)**: it sets the ECHO_PIN as an input.
  - **pinMode(TRIG_PIN, OUTPUT)**: it sets the TRIG_PIN as an output.
  - **message = isOccupied()**: it calls the isOccupied() function.
- It sends the result to the sink node with a transmission power of 2dBm:
  - **WiFi.setTxPower(WIFI_POWER_2dBm)**: it sets the transmission power to 2dBm.
  - **esp_now_send(broadcastAddress, (uint8_t *)&message.c_str(), message.length() + 1)**: it sends the message to the sink node.
- **delay(20)**: it waits 20ms, to ensure that the message is printed correctly after the transmission and reception of the message.
- It powers off the Wi-Fi and enters deep sleep mode for TIME_TO_SLEEP seconds:
  - **WiFi.mode(WIFI_OFF)**: it powers off the Wi-Fi.
  - **esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR)**: it enables the timer wakeup for TIME_TO_SLEEP seconds.
  - **esp_deep_sleep_start()**: it enters deep sleep mode.

Note: in the code provided, it is added a delay of TIME_TO_SLEEP seconds before entering the deep sleep mode, because apparently the simulation on Wokwi is optimized and the deep sleep mode doesn't make the time pass correctly. In a real implementation, this delay should be removed.

2.2.3.2 Loop function

The loop function doesn't do anything relevant, because the node is designed to enter deep sleep mode each time it finishes the setup function.
In the code provided, it simply calls a delay of 10ms.

2.2.4 Example of the output

The output of the code in the first iteration (FREE) is the following:

```
BOOT period -> Completed at [day:0, time:00:00:01.166]
ACTIVE period:
- SENSOR IDLE period -> Completed at [day:0, time:00:00:01.353]
- SENSOR ACTIVE period -> Distance detected: 112.59cm, completed at [day:0, time:00:00:01.361]
- MESSAGE TX period -> Message Sent: OK, completed at [day:0, time:00:00:01.364]
(  [SINK NODE] Receiving Message -> Parking Status: FREE )
Wi-Fi OFF period -> Completed at [day:0, time:00:00:01.390]
Going to sleep now (6.00s)
```

The output of the code in the second iteration (OCCUPIED) is the following:

```
BOOT period -> Completed at [day:0, time:00:00:08.599]
ACTIVE period:
- SENSOR IDLE period -> Completed at [day:0, time:00:00:08.779]
- SENSOR ACTIVE period -> Distance detected: 2.02cm, completed at [day:0, time:00:00:08.781]
- MESSAGE TX period -> Message Sent: OK, completed at [day:0, time:00:00:08.794]
(  [SINK NODE] Receiving Message -> Parking Status: OCCUPIED )
Wi-Fi OFF period -> Completed at [day:0, time:00:00:08.820]
Going to sleep now (6.00s)
```

Note: the received message is printed in the output, between parentheses, but in a real implementation it would be printed by the sink node.

3 Estimation of power and energy consumption

3.1 Datas provided

3.1.1 Power datas provided in CSV files

In order to estimate the power and energy consumption of the node, an average of the datas provided in the set of CSV files is calculated:

deepl_sleep.csv:

- **Deep Sleep**: 59.62mW
- **Boot**: 314.12mW
- **WiFi On**: 776.62mW
- **WiFi Off**: 308.27mW

read_sensor.csv:

- **Sensor Idle**: 331.59mW
- **Sensor Active**: 466.74mW

send_different_TX.csv:

- **TX Idle**: 704.22mW
- **TX 2dBm**: 797.29mW
- **TX 19.5dBm**: 1221.76mW

There are discordant values in the set of CSV files, so the average in divergent cases is calculated, obtaining the following final values:

- **[ESP32] Deep Sleep**: 59.62mW
- **[ESP32] Boot**: 314.12mW
- **[ESP32] WiFi On / TX Idle**: 740.42mW
- **[ESP32] WiFi Off**: 308.27mW
- **[HC-SR04] Sensor Idle**: 331.59mW
- **[HC-SR04] Sensor Active**: 466.74mW
- **[ESP32] TX 2dBm**: 797.29mW
- **[ESP32] TX 19.5dBm**: 1221.76mW

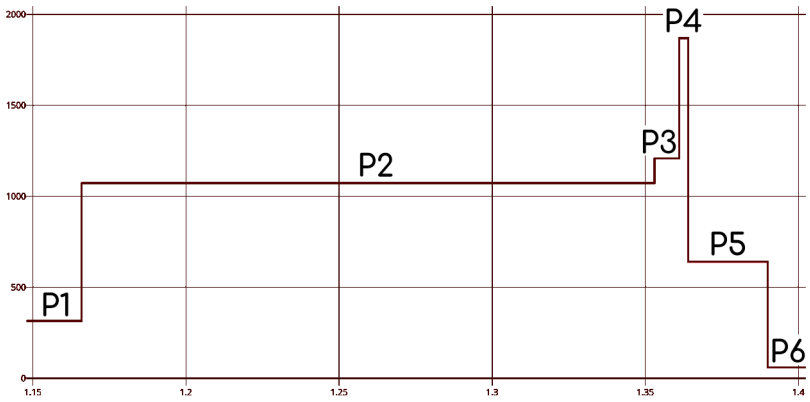
3.1.2 Time datas provided in Wokwi timestamps

The time datas provided in the Wokwi timestamps are the following:

- **[ESP32] Boot**: 1.166s
- **[ESP32] WiFi On**: 1.364s-1.166=0.198s
- **[ESP32] WiFi Off**: 1.390s-1.364=0.026s
- **[ESP32] Deep Sleep**: 6.00s
- **[HC-SR04] Sensor Idle**: (1.353-1.166)+(1.390-1.361)=0.216s
- **[HC-SR04] Sensor Active**: 1.361-1.353=0.008s
- **[ESP32] TX 2dBm**: 1.364-1.361=0.003s
- **[ESP32] TX 19.5dBm**: not used

3.1.3 Power/Time graph

Based on the datas provided, the power graph of 1 iteration/cycle of the node is the following:



Obtained by the following grid:

	Timestamp [s]	Total Power [mW]	ESP32	HC-SR04
P1	0 1.166	314.12 314.12	BOOT	SENSOR OFF
P2	1.166 1.353	1072.01 1072.01	WIFI ON	SENSOR ON (IDLE)
P3	1.353 1.361	1207.16 1207.16	WIFI ON	SENSOR ON (ACTIVE)
P4	1.361 1.364	1869.3 1869.3	WIFI ON + MESSAGE TX 2dBm	SENSOR ON (IDLE)
P5	1.364 1.39	639.86 639.86	WIFI OFF	SENSOR ON (IDLE)
P6	1.39 7.39	59.62 59.62	DEEP SLEEP	SENSOR OFF

3.2 Computation of the power and energy consumption

3.2.1 Average power consumption

Using the grid provided, the power consumption of 1 iteration of the node is estimated with an average of the power states:

- P1: [ESP32] Boot
- P2: [ESP32] WiFi On + [HC-SR04] Sensor Idle
- P3: [ESP32] WiFi On + [HC-SR04] Sensor Active
- P4: [ESP32] WiFi On + [ESP32] TX 2dBm + [HC-SR04] Sensor Idle
- P5: [ESP32] WiFi Off + [HC-SR04] Sensor Idle
- P6: [ESP32] Deep Sleep

For 1 iteration of the node, the average power is the following:

$$P_{avg} = \frac{P1 * 1.166s + P2 * 0.187s + P3 * 0.008s + P4 * 0.003s + P5 * 0.026s + P6 * 6s}{7.390s} = 129.412mW$$

3.2.2 Energy consumption of 1 transmission cycle

So the energy consumption of 1 transmission cycle of the node is the following:

$$E_{avg} = P_{avg} * 7.390s/1000 = 0.956J$$

3.2.3 Battery life

Given a battery of $9451 + 5 = 9456J$, the node can perform $9456J/0.956J \simeq 9891$ iterations.
The battery is estimated to last:

$$\frac{9456J}{0.956} * 7.390s = 73096.067s \simeq \text{20h 18m 16.067s}$$

4 Possible improvements of the sensor node

There are several potential improvements that can be made to reduce energy consumption without modifying the main task of notifying the sink node about the occupancy state of a parking spot. The main ideas are the following:

- **Optimize the deep sleep time**: the deep sleep time can be increased or decreased based on the specific requirements of the application. For example, if the parking spot is in a less busy area, the time can be increased to save energy.
  This can be done also in a **dynamic way**, changing the deep sleep time based on the time of the day or the day of the week (e.g. the deep sleep time can be increased during the night or during the weekend).
- **Optimize the transmission power**: the transmission power can be increased or decreased based on the specific requirements of the application. For example, if the parking spot is in a small area, the power can be decreased to save energy.
  Another idea can also be to change the **topology of the network**, using a different one instead of a star network, in order to reduce the transmission power (e.g. a mesh network).
- **Integrate some form of energy harvesting**: for example, the node can be equipped with a small solar panel if it is placed in an outdoor environment. The solar panel can be used to charge the battery of the node during the day. This can extend the operational lifespan of the node without increasing its energy consumption.

5 Comments and Conclusions

The implemented sensor node is a simple and effective solution to detect the presence of a car in a parking spot and send the information to a central sink node. As stated before there are several potential improvements that can be made to reduce energy consumption without modifying the main task of the node. The estimated battery life of the node is around 20 hours, not very high, but it can be increased with the improvements mentioned before.
The code provided is a simple solution to the problem, but it can be improved in several ways, for example by adding some form of error checking and handling, or by adding some form of data encryption.
In conclusion the project was a good exercise to understand the basics of the ESP32 microcontroller and the ESP-NOW protocol, and to learn how to use the Wokwi platform.