Departamento:

Investigación y Desarollo

Informe:

Placa HELTEC LoRa V3

Versión: 00

Año: 2024

Matias Herrera[1]

[1] *Ingeniería Electrónica*

[*] *Autor correspondiente: matiasherrera2108@gmail.com*

Índice general

1 Conociendo la placa

1.1. Descripción general

En este informe se detalla la placa **WiFi LoRa 32** fabricada por **Heltec**. Es fundamental conocer la versión específica de la placa que se empleará en cualquier proyecto, dado que las características pueden variar significativamente entre diferentes versiones de placas. Este informe está basado en la versión **3** de la placa. Su apariencia es la siguiente:

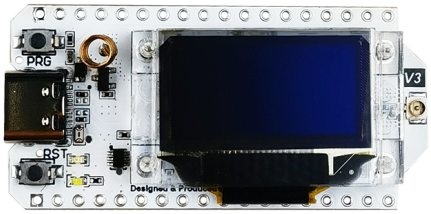

Figura 1.1 – Vista frontal

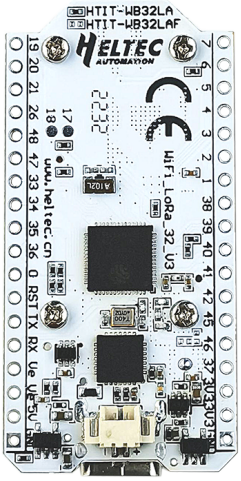

Figura 1.2 – Vista posterior

A continuación se indican las dimensiones de la placa:

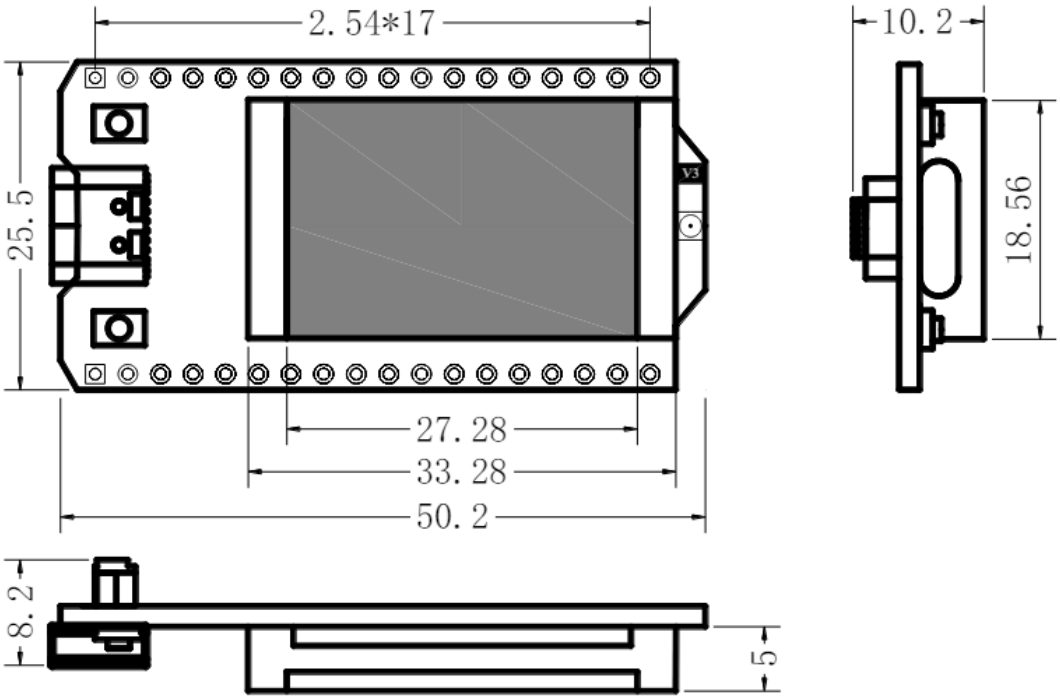

Figura 1.3 – Dimensiones de la placa.

El modulo desarrollado por Heltec ofrece:

- Conexión Wi-Fi.
- Bluetooth 5 (LE).
- Comunicación LoRa.
- Control de carga de bateria integrado.
- Pantalla OLED integrada.
- Pines digitales, ADC y Touch.
- Comunicación UART, I2C y SPI.

1.2. Distribución de pines

La placa posee 36 pines, su distribucion se observa en la figura 1.4, entre ellos tenemos:

- Led blanco incluido en la placa (GPIO 35).
- Pines para medir la tensión de la bateria (GPIO 1 y GPIO 37).
- 9 pines ADC.

- 7 pines Touch.
- 3 puertos para comunicación UART.
- 2 puertos de comunicación I2C.
- 2 puertos para comunicación SPI.
- Más de 20 pines GPIO disponibles para propósitos generales.

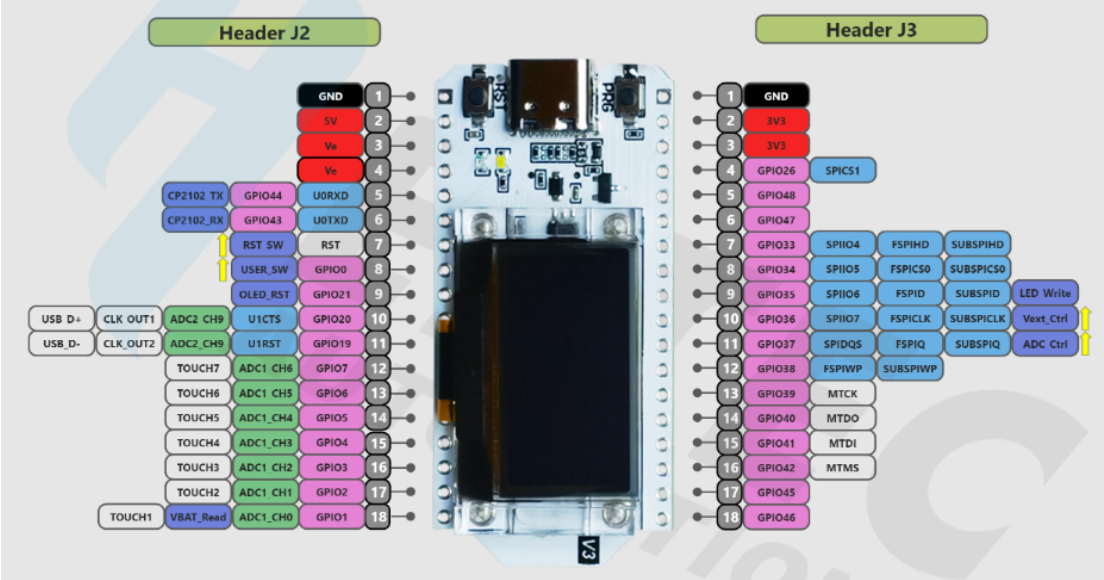

Figura 1.4 – Pinout de la placa.

1.3. Error en serigrafía

Para el proyecto 'CleanTrace' se compraron dos placas Hectec ESP32 LoRa V3 por 'Mercado Libre'. El pinout que nos entrega el fabricante (Figura 1.4) y la distribución de pines según la serigrafía de la placa **no coinciden**. Al momento de realizar un programa para la placa, los pines usados en el firmware deben coincidir con los pines que indica la **serigrafía de la placa**.

1.4. Versiones anteriores

Como se mencionó anteriormente, existen diferentes versiones de la placa **WiFi LoRa 32**. En la siguiente tabla se realiza una comparación entre la version de placa utilizada en este informe (V3) y la version anterior (V2):

	WiFi LoRa 32 (V2)	**WiFi LoRa 32 (V3)**
MCU	ESP32-D0	ESP32-S3
LoRa Chip	SX1276	SX1262
Conector USB	Micro USB	Tipo C
Oscilador de cristal	Estándar	Alta presicion con compensación de temperatura
Consumo en 'sleep mode'	800 uA	<10 uA
Otros	-	Mejor adaptación de impedancia en RF

Una de las diferencias más significativas radica en el **chip LoRa** utilizado. Mientras que el chip **SX1276** (Versión 2) cuenta con numerosos ejemplos que permiten implementar una comunicación LoRa, estos programas no son compatibles con el chip **SX1262**. Se deben buscar librerias especificas para el chip **SX1262**.

En el microcontrolador se pueden observar diferencias importantes en la velocidad de procesamiento. Además, la nueva versión incluye un módulo Bluetooth más reciente. El consumo en modo de reposo ('sleep mode') es considerablemente menor en el nuevo chip. Otro aspecto crucial a tener en cuenta es que la distribución de pines difiere entre la versión anterior y la actual.

2 Software

En este capitulo se presentará un template realizado para ser impementado con la placa Heltec. Su objetivo es facilitar la creacion de nuevos proyectos a partir de un 'código base'. Este template fue desarrollado en Visual Studio Code con la extension platform.io, y se encuentra en un repositorio de GitHub donde se lo puede descargar.

Enlace del repositorio: Heltec WiFi LoRa 32 V3 template

2.1. Aspectos generales

El proyecto tiene la siguiente estructura:

```
/
├── [...]
└── src/
    ├── Adox_Libraries_ESP32/
    │   ├── Bluetooth/
    │   ├── mqtt/
    │   ├── WiFi/
    │   ├── battery_functions.h
    │   ├── EEPROM_functions.h
    │   ├── I2C_scanner.h
    │   ├── oled_esp32.h
    │   ├── RTC_funcions.h
    │   ├── Sensor_MLX90614.h
    │   ├── Serial_functions.h
    │   ├── SPIFFS_functions.h
    │   ├── Timer_tic.h
    │   ├── UDP_functions.h
    │   ├── MFRC522_functions.h
    │   └── gpio_functions.h
    ├── global_variables.h
    └── main.cpp
```

En la carpeta **Adox_Libraries_ESP32** se encuentran todos los módulos de componentes utilizados en el proyecto. Luego el archivo **global_variables.h** contiene variables más generales que son usadas en varios modulos y en el programa principal. El archivo **main.cpp** está destinado al usuario para que realice su programa.

2.2. Módulos

En esta seccion se describirán los modulos más relevantes que se encuentran el la carpeta **Adox_Libraries_ESP32**.

2.2.1. Bluetooth

La placa desarrollada por Heltec tiene integrado un modulo 'Bluetooth 5 LE' (en algunos sitios se lo denomina BLE 5). A continuacion se dara una pequeña introduccion sobre el funcionamiento de este protocolo de comunicacion y como fue implementado en la placa en cuestion.

Con Bluetooth Low Energy (BLE), existen dos tipos de dispositivos: el servidor y el cliente. El ESP32 puede actuar de las dos maneras. Se pueden utilizar dos placas Heltec ESP32, comunicarlas via bluetooth y enviar datos. El servidor anuncia su existencia para que otros dispositivos puedan encontrarlo y contiene datos que el cliente puede leer. El cliente escanea los dispositivos cercanos y, cuando encuentra el servidor que busca, establece una conexión y escucha los datos entrantes. Esto se denomina comunicación punto a punto.
La forma en que dos dispositivos BLE envían y reciben mensajes estándar tiene una estructura definida (Figura 2.1).

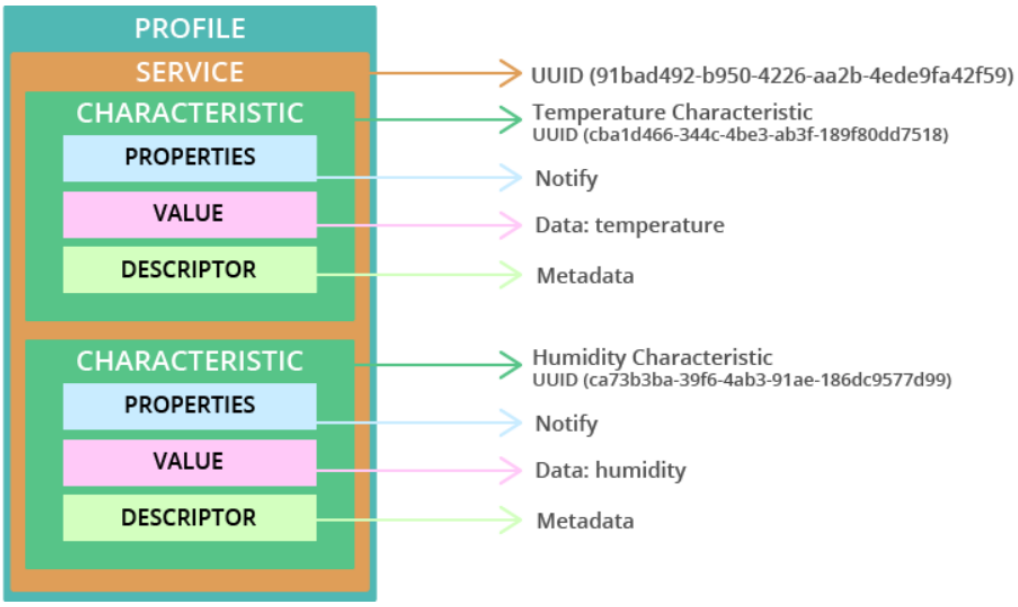

Figura 2.1 – Estructura del protocolo

Al iniciarse un dispositivo BLE como servidor, se crea un **Servicio** con un UUID (identi-

ficador único universal). Dentro de ese 'Servicio' se crean **Características**, destinadas a funciones específicas. En este proyecto se creó una característica para enviar datos (TX) y otra para recibirlos (RX). Cada Característica también tiene su UUID.

Al crear un cliente BLE, debe conectarse al servidor con el UUID del Servicio. Para que el cliente reciba datos del servidor, se debe unir a la Característica TX con el UUID correspondiente. Para enviar datos al servidor, se debe crear una Característica con el UUID de RX.

La estructura del módulo bluetooth creado es la siguiente:

```
Bluetooth
├── Bluetooth_data.h
├── Bluetooth_functions.cpp
├── Bluetooth_client.h
└── Bluetooth_server.cpp
```

Los archivos **Bluetooth_client.h** y **Bluetooth_server.h**, contienen las funciones para el servidor y el cliente, y no es necesario modificarlos.
En **Bluetooth_data.h** se congifuran la UUID del Sevicio, las UUID de las Características y el nombre del servidor. Véase figura 2.2.

```
#include <Arduino.h>
//-------------------- Variables usadas por cliente y servidor:
#define BLE_SERVER_NAME "HeltecBLE"
#define SERVICE_UUID "6E400001-B5A3-F393-E0A9-E50E24DCCA9E"
#define CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"
#define CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"
```

Figura 2.2 – Variables configurables.

En el archivo **Bluetooth_functions.cpp** se elige si la placa Heltec opera como sevidor o como cliente, solamente comentando una línea de código (Véase figura 2.3).

```
#include <Arduino.h>
/**
 *
 * Si se comenta la siguiente linea funciona como cliente.
 * Si NO se comenta la siguiente linea funciona como server.
 *
 */
#define SERVER_CODE
```

Figura 2.3 – Configurar modo cliente o servidor.

El el programa principal (Figura 2.4) se debe incluir la carpeta **Bluetooth**. Es importante tener en cuenta que las funciones y variables que se nombrarán a continuación son independientes del modo BLE de la placa, es decir, si la placa se usa como cliente o servidor (Figura 2.3). En el 'setup()' se debe invocar a la función **BluetoothBegin()** para iniciar la comunicación.
En el 'loop()' se debe utilizar:

- **BluetoothLoop**: esta función se encarga de comprobar si hay nuevos mensajes y verificar la conexión. Si la placa está en modo servidor, verifica que el cliente esté conectado a él; si la placa funciona como cliente, verifica su conexión al servidor.
- **ble_status**: esta es una variable booleana. Si se encuentra en true, significa que la conexión está establecida; si se encuentra en false, significa que el dispositivo BLE está desconectado.
- **BluetoothSend**: función que recibe el String que se desea enviar. Si trabajamos en modo cliente, envía los datos al servidor; si la placa está en modo servidor, se envían los datos al cliente.
- **ble_flag_new_data**: variable booleana para chequear si se recibieron nuevos mensajes. Si se encuentra en true, se recibió un mensaje nuevo; si se encuentra en false, no existen nuevos mensajes.
- **ble_msg**: esta variable tipo String contiene el mensaje recibido.
- **ble_mode**: esta variable es un String que indica en qué modo está operando la placa: cliente o servidor.

```ino
#include "Adox_Libraries_ESP32/Bluetooh/Bluetooth_functions.h"

void setup()
{
    BluetoothBegin(); // Iniciar bluetooth.
}

void loop()
{

    BluetoothLoop(); // Recibir mensajes BLE y comprobar conexión.

    //---- Estado de la conexión BLE:
    //---- ble_status = true -> Conectado
    //---- ble_status = false -> Desconectado
    if (ble_status)
    {
        Serial.print("BLE conectado");
    }
    else
    {
        Serial.print("BLE desconectado");
    }


    //---- Para enviar datos:
    String data = "Test_msg";
    BluetoothSend(data); //Envío "data" vía BLE.


    //---- Recepción de nuevos mensajes.
    // ble_flag_new_data = true; -> Nuevo mensaje recibido.
    // ble_mode: indica el modo de trabajo de la placa (client o servidor)
    if (ble_flag_new_data)
    {
        Serial.println("Se recibió un mensaje nuevo");
        Serial.print(ble_mode + ble_msg);
    }
}
```

Figura 2.4 – Operando BLE en el programa principal.

2.2.2. mqtt

Este módulo tiene la siguiente estructura:

```
mqtt
├── MqttData.h
├── MqttLibrary.h
└── MqttLibrary.cpp
```

Se intentó armar una especie de librería para simplificar el uso de la tecnología MQTT. Los archivos **MqttLibrary.h** y **MqttLibrary.cpp** contienen las variables, clases y funciones utilizadas. En el archivo **MqttData.h** (Figura 2.5) se pueden configurar los datos del servidor MQTT, nombre del usuario y prefijo de cada cliente MQTT.

```
#define topic_prefix "ESP32_Heltec" //* Group name */
#define mqtt_default_user "adoxdesarrollos" //* Mqtt server user */
#define mqtt_default_password "Adox2311"//* Mqtt server password */
#define mqtt_server_ip "143.198.229.75" //* Mqtt server ip address */
#define mqtt_reconnect_time 500 //* Time to reconnect */
//#define serial_debug //* Enable serial debug */
```

Figura 2.5 – Archivo MqttData.h

En el archivo **main.cpp** (2.6) se debe utilizar del siguiente modo:

```
#include "Adox_Libraries_ESP32/mqtt/MqttLibrary.h"
mqtt_wifi mq; // Objeto

void setup()
{
    mq.begin();
    mq.set_wifi(ssid, dir_ssid, pass, dir_pass);
    mq.set_datos_set_time(&datos_set, dir_datos_set);
    mq.set_topic_dir(dir_mqtt_topic); // Enviar dirección EEPROM del topic.
}
void loop()
{
    mq.loop(); // Peticiones de mqtt
    //----- Enviar datos por MQTT:
    String dato = "Mensaje de prueba";
    mq.send(dato); // Se envía 'dato' al server MQTT.
}
```

Figura 2.6 – Como operar en main.cpp

Función set_wifi: a esta función se le deben pasar las direcciones de las variables **ssid** (red WiFi) y **pass** (red WiFi). De esta manera cuando se reciben mensajes de configuración del server MQTT se interpretan internamente, se modifican éstas variables directamente, también se las guarda en la memoria EEPROM, y desde el **main.cpp** no debe implementarse ningún código. La función **set_wifi** recibe cuatro (4) parámetros:

- ssid: variable tipo char[ ].
- dir_ssid: dirección de la memoria EEPROM donde se guarda la variable 'ssid'.
- pass: variable tipo char[ ].
- dis_pass: dirección de la memoria EEPROM donde se guarda la variable 'pass'.

De la misma manera existe la función **set_datos_set_tim** para la variable **datos_set** (utilizada para configurar los intervalos de envio de datos al servidor 'AmbienteControlado')

2.2.3. WiFi

El contenido de este módulo es similar al que se venía utilizando en proyectos anteriores con la placa **ESP8266**. Solo contiene algunas modificaciones para la compatibilidad con la placa Heltec. En el **main.cpp** se debe emplear así:

```
#include "Adox_Libraries_ESP32/WiFi/WiFi_functions.h"

void setup()
{
    ESP32_setup_wifi(); // Conexión a la red WiFi.
    ESP32_modoconf();   // Configuración del WebServer.
}
void loop()
{
    ESP32_loop(); // Recibe peticiones de WiFi.
}
```

Figura 2.7 – Como usar WiFi en main.cpp

2.2.4. battery_functions.h

Cuando se alimenta la placa Heltec ESP32 con una batería en su conector dedicado (Figura 2.8), se desea conocer el estado de la misma. La placa posee dos pines destinados a conocer el estado de la batería, estos pines son: **Pin 37: 'ADC_Ctrl'** y **Pin 1: 'VBat_Read'**. Dentro del archivo **battery_functions.h** se encuentran las referencias a estos pines y las configuraciones de los mismos.
En el archivo main.cpp bastará con invocar una función para configurar los pines, otra

para realizar la lectura y operar las variables **battery_value** y **battery_percentage** directamente. Véase figura 2.9

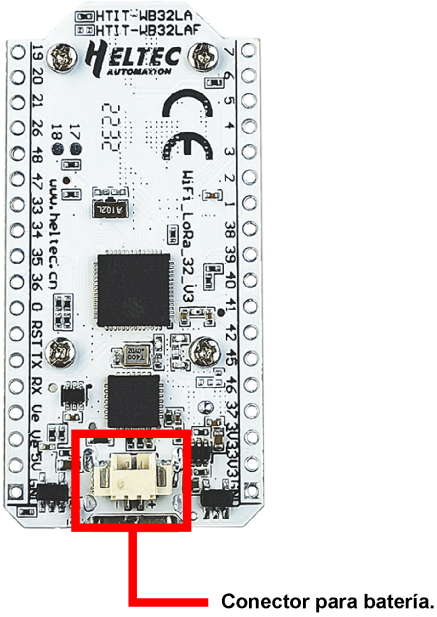

Figura 2.8 – Conector para batería.

```
#include "Adox_Libraries_ESP32/battery_functions.h"

void setup()
{
    battery_config(); // Configuración de los pines utilizados.
}

void loop()
{
    battery_read(); // Leer la tensión de la bateria y el porcentaje.

    //----> Mostrar datos de la batería.
    Serial.println("Tensión: " + String(battery_value) + " V.");
    Serial.println("Porcentaje: " + String(battery_percentage) + " %.");
}
```

Figura 2.9 – Mostrar la tensión y porcentaje de carga de la batería.

2.2.5. MFRC522_functions.h

Éste módulo permite la lectura de RFID desde el dispositivo **MFRC522**. La figura 2.10 muestra las conexiones entre el lector de RFID con la placa Heltec.

En el programa principal (main.cpp) deben invocarse dos funciones y operar sobre dos variables como se muestra en la figura 2.11

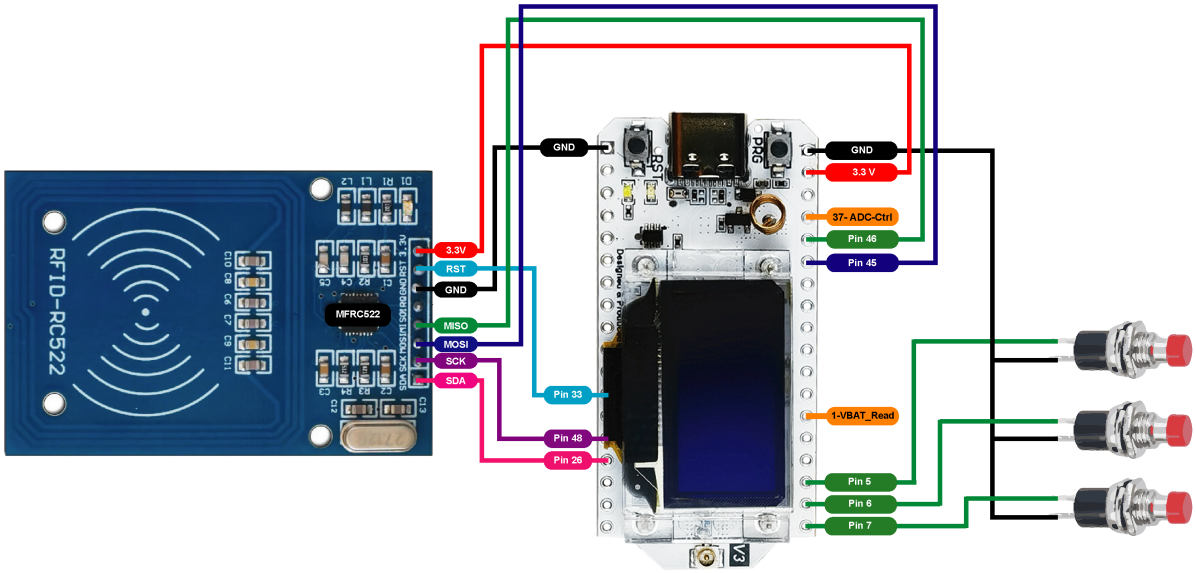

Figura 2.10 – Diagrama de conexiones

```cpp
#include "Adox_Libraries_ESP32/MFRC522_functions.h"

void setup()
{

    MFRC522_begin(); // Inicialización del módulo.
}

void loop()
{

    MFRC522_loop(); // Lectura de tarjetas RFID.

    //---- Cuando se lee una tarjeta RFID se activa "flag_new_rfid" = true
    if (flag_new_rfid)
    {
        flag_new_rfid = false;
        Serial.println("ID de la tarjeta RFID: " + rfid_serial);
    }
}
```

Figura 2.11 – En el archivo main.cpp

3 Próximas mejoras

3.1. Sistema de archivos SPIFFS

Se debe implementar el manejo del sistema de archivos SPIFFS de la placa Heltec. Esto permitirá almacenar archivos al estilo datalogger y acceder a ellos desde **ip/download**, como se ha hecho en proyectos anteriores.

3.2. Control de RTC

Se deben desarrollar las funciones para gestionar el reloj RTC, asegurando que la fecha y hora estén siempre actualizadas. Esto es crucial para el envío de datos y el funcionamiento del datalogger.