UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

"PLC PARA CONTROL DE BOMBAS DE PISCINA Y SISTEMA DE RIEGO AUTOMÁTICO EN CHALÉS"

TRABAJO FIN DE GRADO

Septiembre – 2020

Autor: Juan Luis Leal Contreras

Director: David Marroquí Sempere

Resumen

El objetivo de este trabajo de fin de grado es diseñar y fabricar un autómata para el control de bombas de piscina y de riego así como electroválvulas. Dispondrá de ocho salidas de relé, dos de ellas dedicadas para motores (hasta 17,5 A @250 V) y seis para uso genérico (electroválvulas, tomas de corriente, etc.) (hasta 5 A @250 V). El control de estas salidas se realizará desde cualquier dispositivo con conexión wifi mediante una página web desde la cual se podrá encender, apagar y programar las ocho salidas de forma semanal y anual con intervalos de diarios de media hora. La programación de las salidas se almacenará en la memoria interna del microcontrolador de forma que esta no se pierda en caso de que se retire la alimentación.

El autómata estará dividido en dos módulos: el primer módulo actuará de servidor y es el que se conectará a la red wifi y generará la página web, y el segundo módulo estará situado allí donde se deseen automatizar las salidas. La comunicación entre ambos módulos será inalámbrica y se realizará mediante módulos de radiofrecuencia los cuales permiten un alcance de varios kilómetros.

Para mantener la fecha y hora, el autómata cuenta con un reloj interno y una pila de botón.

Desde la página web generada por el autómata se podrá también consultar la temperatura del agua de la piscina, así como la temperatura del propio módulo, la calidad de la conexión entre ambos módulos y la fecha y hora del reloj.

Palabras clave

PLC, autómata, LoRa, ESP32, relé, PCB, web.

ÍNDICE

1. INTRODUCCIÓN

1.1. MOTIVACIÓN

Muchos chalés y urbanizaciones disponen de piscina y jardines. En la mayoría de casos el control de los mandos tanto de la piscina como del riego se encuentra situado cerca de las bombas y por lo tanto para accionarlas es necesario desplazarse hasta el cuadro de mando y accionar allí los interruptores.

Hoy en día hay muchas soluciones para la automatización tanto de riego como del control de bombas de piscina pero suelen ser PLCs (*programmable logic controller*, controlador lógico programable) muy caros y/o poco flexibles. A continuación mostraremos algunas de las soluciones que podemos encontrar actualmente en el mercado:

- **Reloj programador analógico**

  Este tipo de programador es muy común para el encendido de bombas de piscina o de cualquier salida que queramos que se encienda en un horario determinado. Suele estar compuesto por una serie de pines situados alrededor de un círculo en el que cada pin representa una unidad de tiempo, normalmente 30 o 15 minutos. Cuando la aguja del reloj pasa por encima de un pin activado la salida se enciende.

  Este tipo de programador es muy económico y es por ello que es muy común encontrarlo en la mayoría cuadros eléctricos de piscinas pero está muy limitado ya que solamente dispone de tres posiciones, encendido, apagado y reloj, las cuales deben ser accionadas manualmente. Además la programación es diaria, por lo que no nos servirá si queremos que la salida se encienda en días alternados o únicamente durante los meses que dure la temporada de baño.

- **Reloj programador digital**

Este tipo de programador es muy similar al reloj analógico pero tiene tres ventajas principales:

  - No sufre deterioro físico durante la programación ya que no dispone de pines.
  - Suele disponer de varios programas de tal forma que se puede alternar de uno a otro.
  - Cada programa suele ser semanal y por lo tanto no queda limitado únicamente a 24 horas.

A pesar de las ventajas respecto al analógico, sus programas siguen siendo bastante limitados al ser semanales, no pudiendo ser programado de forma mensual o anual. Además para su programación se requiere que el usuario esté presente en la sala donde esté el cuadro eléctrico.

Figura 1: Reloj programador analógico

Figura 2: Reloj programador digital

- **Autómata o PLC**

Los autómatas o PLCs son dispositivos electrónicos que se utilizan para automatizar procesos industriales. Están diseñados para procesar múltiples señales de entrada y de salida, y al estar pensados para entornos industriales, suelen contar con inmunidad al ruido eléctrico y resistencia a la vibración y al impacto [1].

Un PLC es una solución bastante cara: el precio medio ronda los 200 € [2], mientras que el precio medio del reloj programador analógico ronda los 20 € [3] y el digital los 40 € [4].

Además el uso un PLC par la automatización que pretendemos llevar a cabo desaprovecharía las capacidades que tiene este dispositivo, ya que cuenta con varias entradas para sensores que no se utilizarían. Estos se suelen emplear para la automatización integral de una vivienda, pero no únicamente para el sistema de riego o las bombas de la piscina.

A pesar de ser una solución muy fiable sigue siendo poco flexible, ya que para modificar su funcionamiento es necesario reprogramarlo. Además el autómata por sí solo no es suficiente, se necesita un módulo de relés o un contactor para efectuar el encendido o apagado del circuito de potencia (bombas, electroválvulas, enchufes, etc.).

Figura 3: PLC Siemens S7-1200

Otros productos como por ejemplo las estaciones de riego automático siguen siendo caras (precio medio de 150 € [5]) y únicamente sirven para el accionamiento de electroválvulas y no para salidas de potencia en general.

En definitiva, los usuarios cada vez demandan más productos para la automatización de sus viviendas y que a la vez sean fáciles de manejar.

1.2. OBJETIVOS

Es por ello que se ha decidido desarrollar un PLC que reúna la siguientes características:

- Controlable y programable desde una página web (a través de wifi).
- Que integre ocho salidas de relé (sin necesidad de contactores externos) capaces de gobernar 5 A @ 250 VAC para las salidas genéricas y 17,5 A @ 250 VAC para las dedicadas para motores.
- Preparado para ambientes húmedos.
- Alcance de la zona a automatizar de varios kilómetros de forma inalámbrica.
- Bajo coste (menos de 100 €).
- Bajo consumo eléctrico (menos de 5 W).

2. DESCRIPCIÓN DEL PRODUCTO

2.1. DESCRIPCIÓN GENERAL DEL PRODUCTO

Para solucionar el problema de la flexibilidad a la hora de manejar y programar el encendido y apagado de bombas de piscina, de riego y en definitiva de cualquier salida de potencia, se ha desarrollado, diseñado y fabricado un PLC controlable a través de una página web desde la cual se puede programar y accionar ocho relés: dos para el accionamiento de motores (hasta 17,5A @250V) y seis de uso genérico (electroválvulas, tomas de corriente, etc.) (hasta 5A @250V).

Ya que uno de los objetivos es que el dispositivo se pueda manejar a través de un smartphone mediante una web, es necesario que el PLC se encuentre conectado a un rúter. Esta conexión será hará mediante wifi por los motivos especificados en el apartado 3.3.3.

En este tipo de instalación, en general cuadro de mandos de bombas de piscina o de riego suele estar situado en el exterior de la vivienda donde la calidad de la conexión wifi suele ser muy baja y en muchos casos inexistente, se propone que el PLC esté dividido en dos módulos: uno situado en el interior de la vivienda y otro cerca del cuadro de mando de la piscina. A estos módulos los llamaremos a partir de ahora Módulo Servidor y Módulo de Relés respectivamente.

La comunicación entre ambos módulos se realiza mediante radiofrecuencia con módulos LoRa (Long-Range), facilitando así la instalación. Además esta conexión permite alcanzar distancias de hasta 15 kilómetros [6], asegurando que aunque haya obstáculos entre ambos módulos tales como tabiques o incluso instalaciones en sótanos, la conexión esté garantizada.

Estas son las características más importantes de cada módulo:

- **Módulo Servidor**

  Estará situado cerca del rúter wifi de tal forma que pueda conectarse a esta red y actuar como servidor de una página web desde la cual se puedan controlar y programar las ocho salidas de potencia además de comprobar su estado (encendido, apagado o en programa) y conocer información adicional como la hora y la temperatura del Módulo de Relés (termómetro interno del reloj incluido en este módulo), la temperatura del agua de la piscina medida a través de una sonda (opcional) y la calidad de la conexión entre los módulos.

Figura 4: Módulo Servidor

Figura 5: Módulo Servidor (reverso)

- **Módulo de Relés**

Estará situado en el lugar donde se desean automatizar las salidas y dispondrá de ocho relés: seis genéricos (5A @250VAC) y dos dedicados para motores (30A @250VAC). Para mantener la hora y fecha dispone del RTC (*real time clock*, reloj en tiempo real) DS3231M [7] junto con una pila de botón (tamaño 1220). Para el accionamiento de las salidas dispone del integrado TLE 8108 EM [8] de la marca Infineon, el cual es capaz de controlar hasta ocho relés. Además dispone también de ocho conectores de dos pines (uno para cada salida) y un conector adicional de tres pines que servirá para conectar una sonda de temperatura con el sensor DS18B20 [9], la cual servirá para medir la temperatura del agua de la piscina.

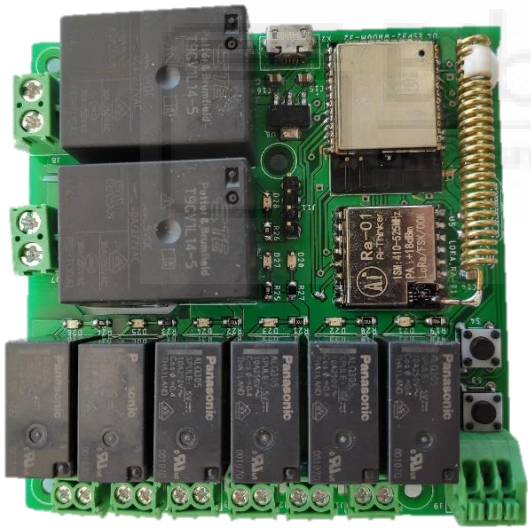


Figura 6: Módulo de Relés

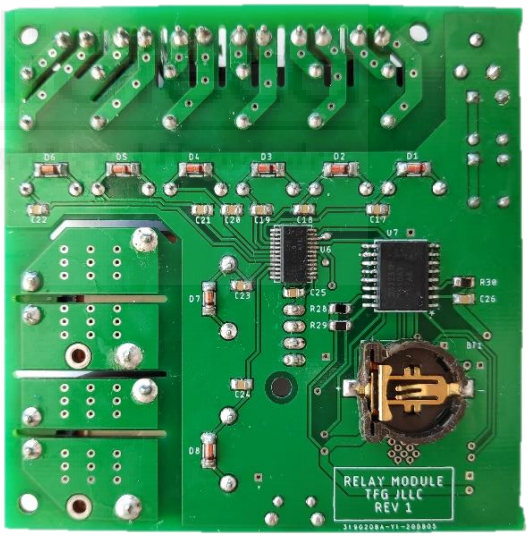


Figura 7: Módulo de Relés (reverso)

Ambos módulos se alimentan a 5VDC mediante un conector micro USB de forma que pueda aprovechar los cargadores de móviles o fuente de alimentación de dispositivos electrónicos que funcionen a este voltaje.

Como microcontrolador cada módulo dispone de un ESP32 WROOM 32 de la marca Espressif, el cual integra una antena de wifi en la propia placa y ha demostrado ser perfecto para proyectos IoT (*internet of things*, internet de las cosas) [10].

La conexión entre los módulos Servidor y de Relés se realiza mediante módulos LoRa Ra-01 [11] de la marca Ai-Thinker, los cuales funcionan en la banda de los 433MHz y debido a su largo alcance y su reducido precio y son ideales para proyectos que requieran transmisión de datos a largas distancias [6].

Además cada módulo dispone de varios diodos led:

- Un led azul para indicar si hay alimentación.
- Un led rojo para indicar que se está transmitiendo o recibiendo algún mensaje mediante el módulo LoRa.
- Ocho diodos led de color verde, uno por cada salida, los cuales indican si la salida está activada o desactivada.

2.2. DIAGRAMA DE BLOQUES

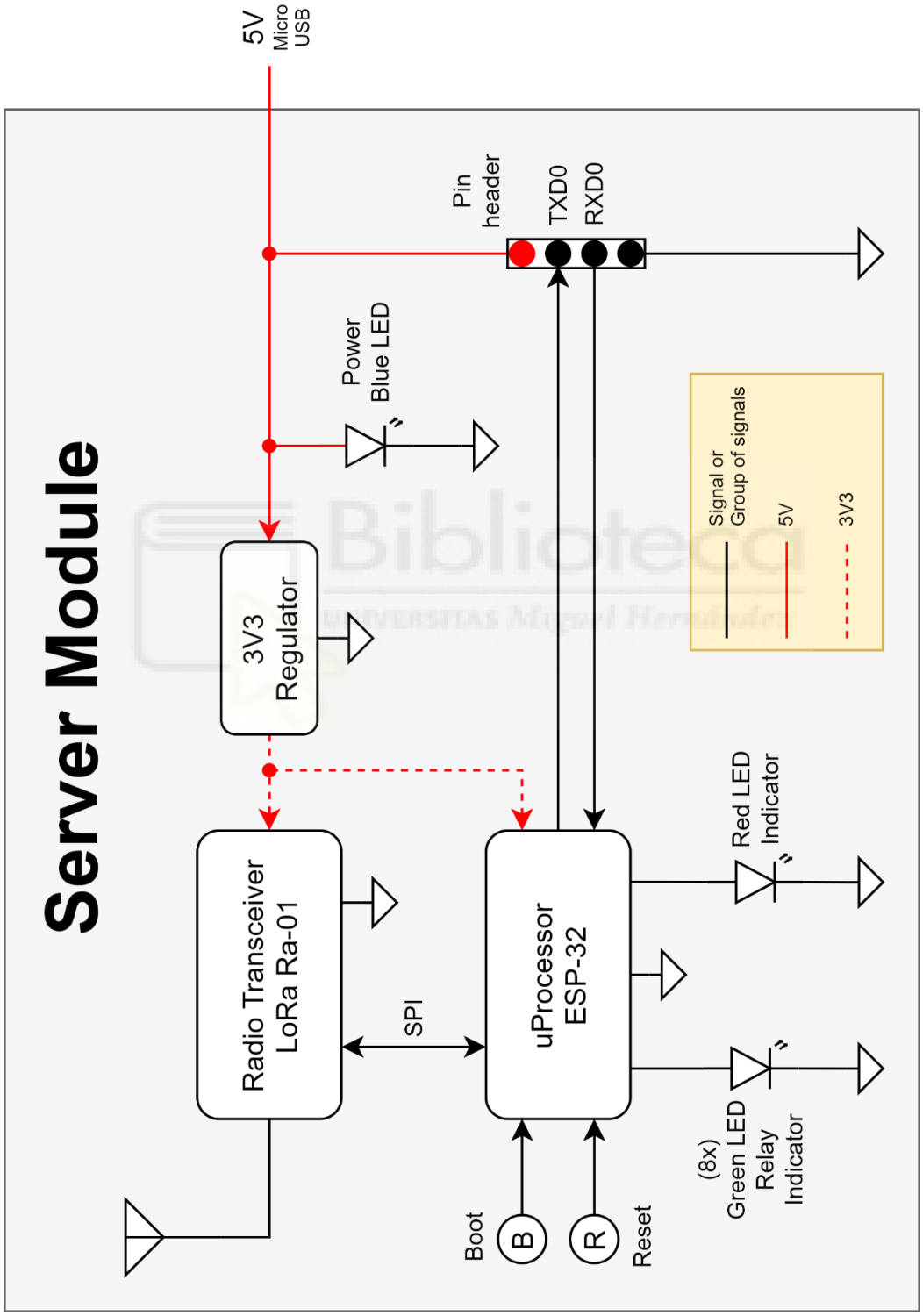


Figura 8: Diagrama de funcionamiento del Módulo Servidor

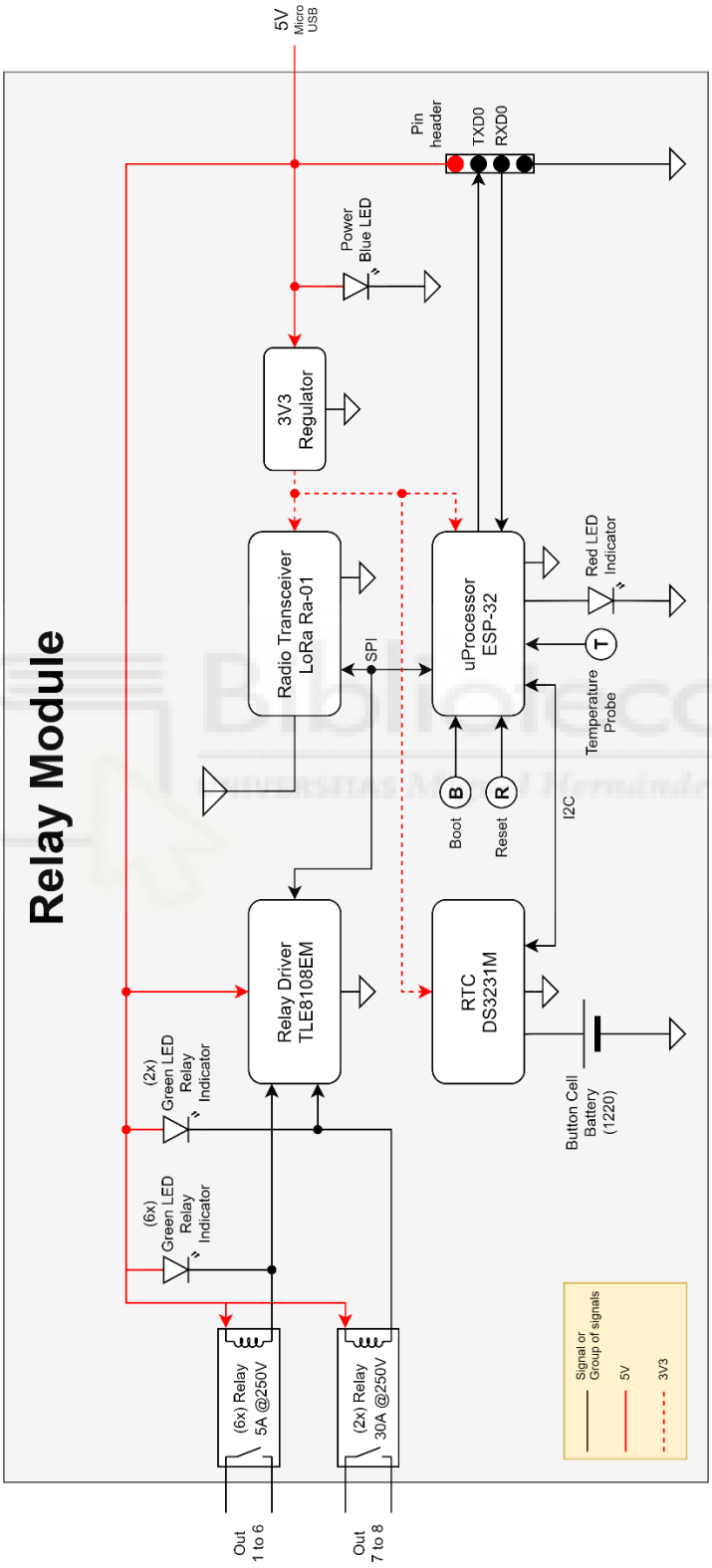


Figura 9: Diagrama de funcionamiento del Módulo de Relés

2.3. DESCRIPCIÓN FÍSICA

Ambos módulos han sido diseñados para ocupar el mínimo espacio posible, integrando en la propia PCB tanto la antena wifi (incluida en el SoC ESP32 WROOM 32 [10]) como la antena de radiofrecuencia de los módulos LoRa Ra-01.

La antena de radiofrecuencia consta de un alambre enrollado en forma de muelle que se sitúa de forma paralela a la PCB. De esta manera no sobresale de la placa ni aumenta de forma considerable la altura de los módulos. Al tratarse de un muelle de metal desnudo y por lo tanto conductor de la electricidad, este irá pegado a la placa de forma que sobresalga unos milímetros por encima de los componentes y no puedan producirse cortocircuitos debidos a las vibraciones de la antena.

Figura 10: Antena de radiofrecuencia para 433MHz

Figura 11: Detalle de la sujeción de la antena

- **Módulo Servidor**

  El tamaño de la PCB de este módulo es de 40x80 mm y la altura teniendo en cuenta los componentes ya montados en la placa es de 11 mm.

  Con el fin de proteger la PCB y sus componentes, el módulo irá resguardado en una caja de plástico (caja de plástico no realizada). La caja deberá incluir aberturas para todos los led de la PCB, así como dos agujeros de 1,3 cm de diámetro en la parte superior para el accionamiento de los botones incluidos en la placa, una ranura para el *pin header* (puerto serie) de programación del microcontrolador y otra ranura para la entrada micro USB para alimentación. El módulo irá fijado a este caja mediante tres tornillos.

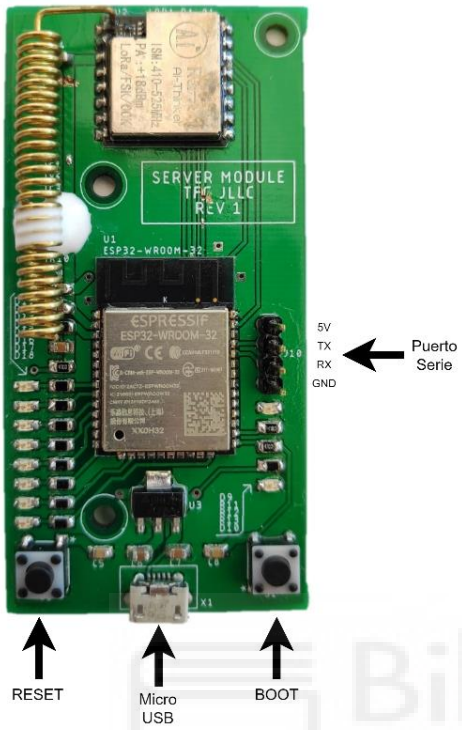


Figura 12: Partes del Módulo Servidor

Figura 13: Concepto de caja de plástico protectora. Diseño propio realizado en Inventor

- **Módulo de Relés**

El tamaño de la PCB de este módulo es de 80x80 mm y la altura teniendo en cuenta los componentes ya montados en la placa es de 20 mm.

Debido a la finalidad que se le va a dar a este módulo el cual estará situado en ambientes húmedos o a la intemperie (sala de máquinas de piscinas o similar), el módulo irá protegido en el interior de una caja de conexiones estanca, en la cual se realizarán las conexiones necesarias y se pasarán los cables por los orificios dedicados para ello. Se recomienda que las medidas de esta caja sean de 85x85 mm. El módulo irá fijado a la caja de conexiones mediante cuatro tornillos.

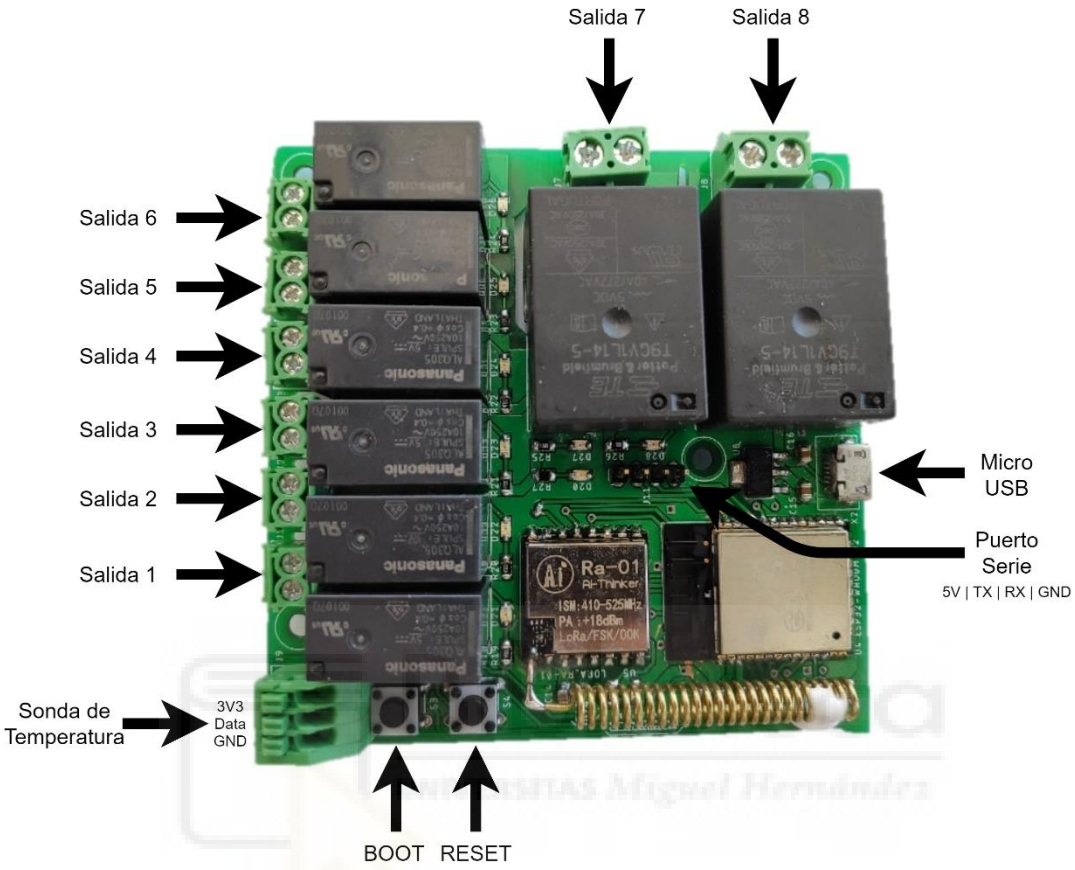


Figura 14: Partes del Módulo de Relés

Figura 15: Caja estanca de conexiones [12]

2.4. ESPECIFICACIONES TÉCNICAS

2.4.1. CARACTERÍSTICAS GENERALES

- **Peso**

  El peso del módulo servidor es de 18,4 g (sin contar caja de plástico) y el peso del módulo de relés es de 118,0 g.

- **Calidad de la conexión wifi**

  El alcance de la conexión wifi en condiciones normales (sin obstáculos entre el módulo servidor y el rúter) es de 20 m. Sin embargo se recomienda mantenerlo a menos de 5 m del rúter, para que la calidad de conexión sea óptima. Para más información referirse al apartado 5.2.

- **Calidad de la conexión LoRa**

  Se recomienda mantener los módulos a menos de 1000 m de distancia y a menos de 100 m si hay gran cantidad de obstáculos, como por ejemplo si alguno de los módulos se encuentra en un sótano. Para más información referirse al apartado 5.1.

  En el siguiente gráfico podemos observar como la calidad de la conexión disminuye según la distancia (línea recta y en ausencia de obstáculos):

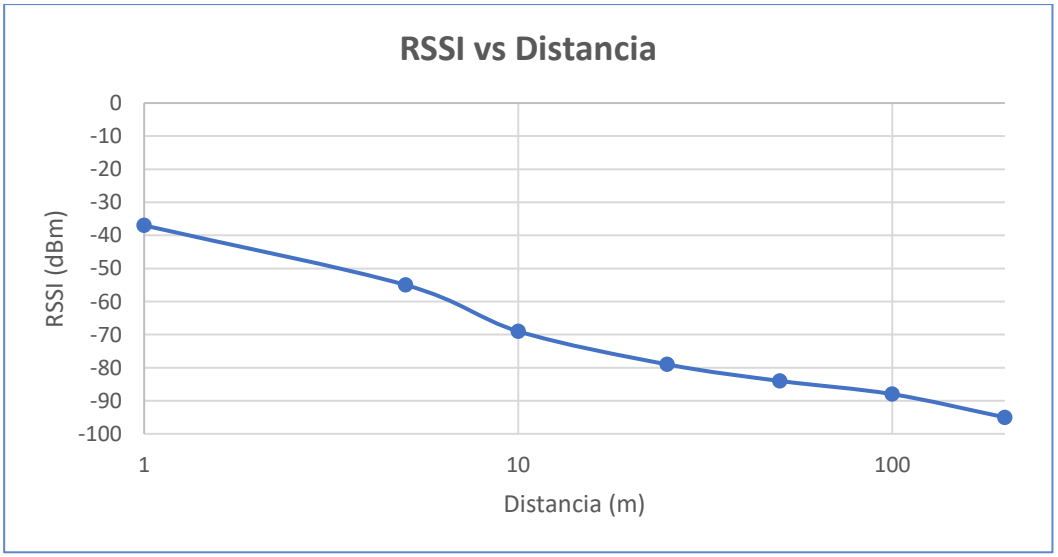


Figura 16: Gráfico de la calidad de la conexión LoRa

2.4.2. CARACTERÍSTICAS ELÉCTRICAS

- **Módulo Servidor**

Parámetro	**Símbolo**	**Valor**	**Unidad**
Voltaje de alimentación	V_{in}	5	V
Voltaje del microcontrolador	$V_{\mu C}$	3.3	V
Frecuencia de radio LoRa[1)]	f_{LoRa}	433	MHz
Frecuencia de reloj SPI[1)]	f_{SPI}	1	MHz
Consumo en reposo	$P_{S,IDLE}$	375	mW
Consumo adicional por cada led verde	P_{GLED}	22	mW
Consumo adicional al transmitir	P_{Tx}	500	mW

- **Módulo de Relés**

Parámetro	**Símbolo**	**Valor**	**Unidad**
Voltaje de alimentación	V_{in}	5	V
Voltaje del microcontrolador	$V_{\mu C}$	3.3	V
Voltaje de bobina de relés	V_{relay}	5	V
Frecuencia de radio LoRa[1)]	f_{LoRa}	433	MHz
Frecuencia de reloj SPI[1)]	f_{SPI}	1	MHz
Consumo en reposo	$P_{R,IDLE}$	375	mW
Consumo adicional por cada led verde	P_{GLED}	22	mW
Consumo adicional por cada relé genérico	$P_{GpRelay}$	200	mW
Consumo adicional por cada relé para motores	P_{MRelay}	900	mW
Consumo adicional al transmitir	P_{Tx}	500	mW
Corriente máxima de conmutación, salidas 1 a 6	$I_{1-6,max}$	10	A
Voltaje máximo de conmutación, salidas 1 a 6	$V_{1-6,max}$	250	$V\ AC$

Capacidad de conmutación nominal, salidas 1 a 6	–	5 @30VDC Resistive 5 @250VAC Resistive 10 @125VAC Resistive	A
Ciclos de vida los contactos, salidas 1 a 6	–	10^5@5A, 30VDC $5 \cdot 10^4$@5A, 250VAC $5 \cdot 10^4$ @10A, 125VAC	Ciclos
Corriente máxima de conmutación, salidas 7 a 8	$I_{7-8,max}$	17,5	A
Voltaje máximo de conmutación, salidas 7 a 8	$V_{7-8,max}$	480	$V\ AC$
Capacidad de conmutación nominal, salidas 7 a 8	–	30 @250VAC Resistive	A
Carga mínima recomendada, salidas 7 a 8	–	1 @12VAC/VDC	A
Ciclos de vida los contactos, salidas 7 a 8	–	10^5 @30A, 277VAC, General Purpose, 85°C	Ciclos

1) Tanto la frecuencia SPI como la de transmisión de los módulos LoRa son modificables a través de código.

Todas las salidas de relé son normalmente abiertas [13], [14].

Para más información sobre el consumo de los módulos referirse al apartado 5.3.

3. HARDWARE

La comunicación entre el microcontrolador ESP32 y el módulo LoRa se realiza mediante el protocolo SPI (*Serial Peripheral Interface*) en el que el microcontrolador ESP32 actúa como maestro y el módulo LoRa como esclavo. En el caso del Módulo de Relés, el driver TLE 8108 EM actúa también como esclavo.

La frecuencia de la señal de reloj SPI es de 1 MHz, modificable a través del código. La frecuencia máxima está limitada por el driver de los relés a un máximo de 5 MHz.

La comunicación entre el microcontrolador ESP32 y el RTC DS3231M se realiza mediante el protocolo I2C.

3.1. ESQUEMÁTICOS

Los esquemáticos de ambos módulos los podemos encontrar íntegros en el Anexo II.

3.1.1. ESQUEMÁTICO DEL MÓDULO SERVIDOR

- **Alimentación**

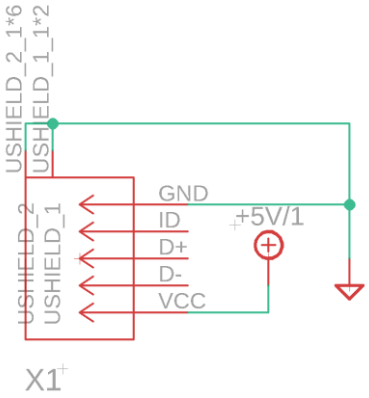


Figura 17: Micro USB (esquemático Módulo Servidor)

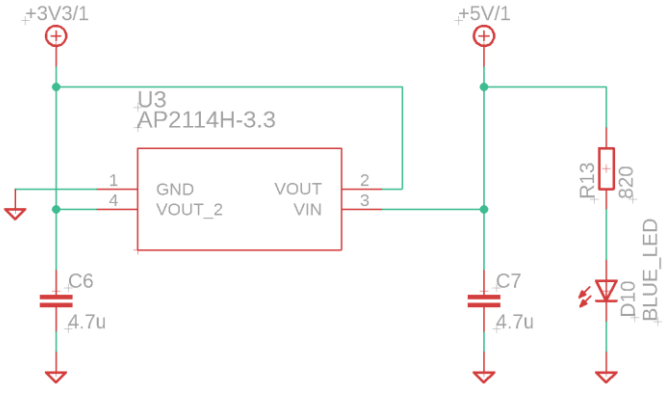


Figura 18: Regulador lineal 5V a 3V3 (esquemático Módulo Servidor)

- **Microcontrolador**

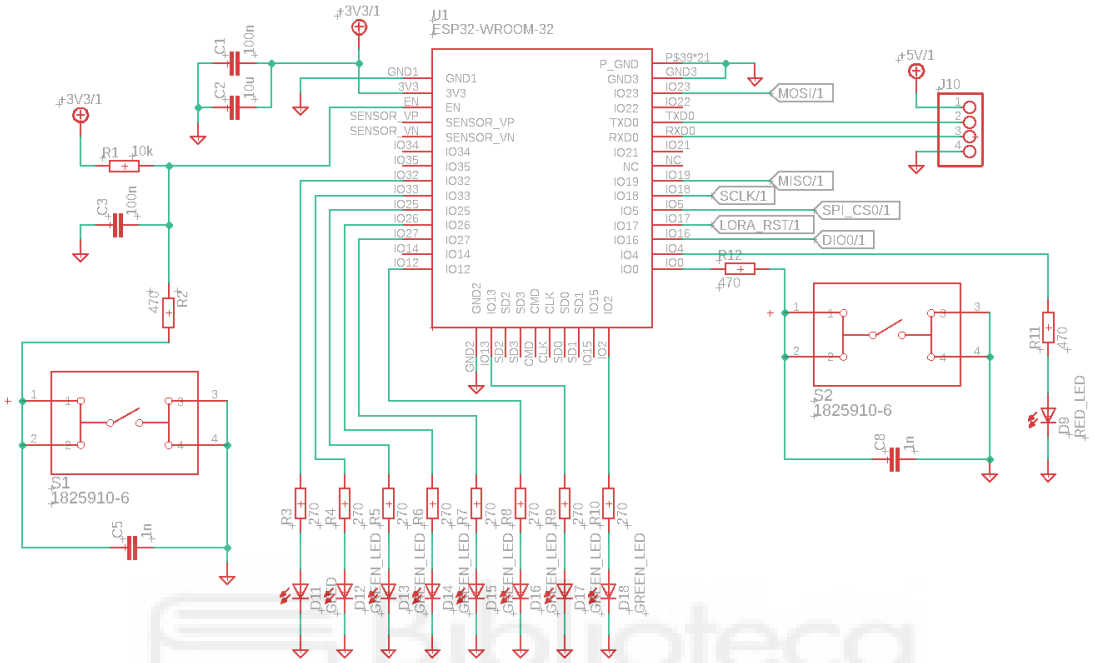


Figura 19: Microcontrolador y periféricos (esquemático Módulo Servidor)

- **Módulo LoRa**

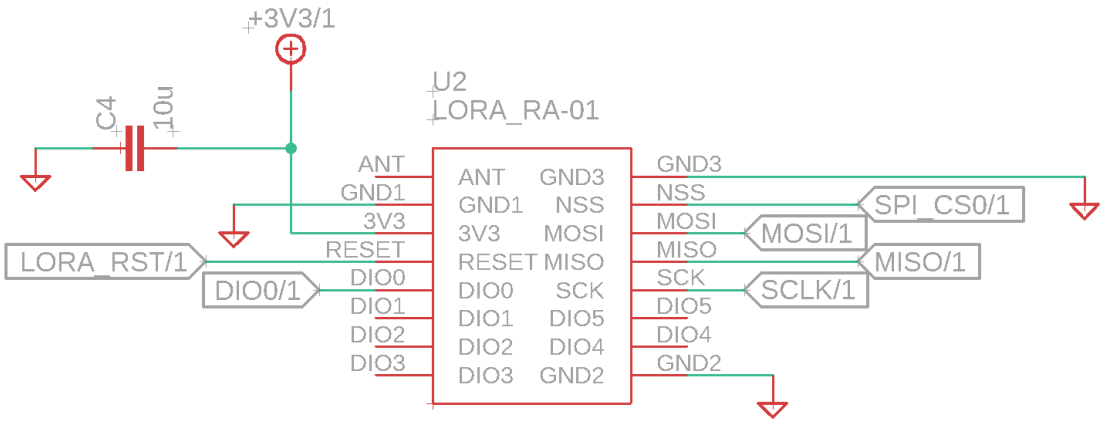


Figura 20: Módulo de radiofrecuencia LoRa (esquemático Módulo Servidor)

3.1.2. ESQUEMÁTICO DEL MÓDULO DE RELÉS

- **Alimentación**

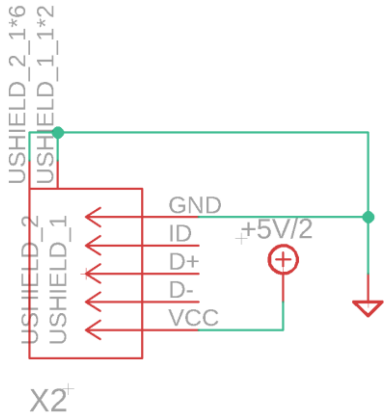


Figura 21: Micro USB (esquemático Módulo de Relés)

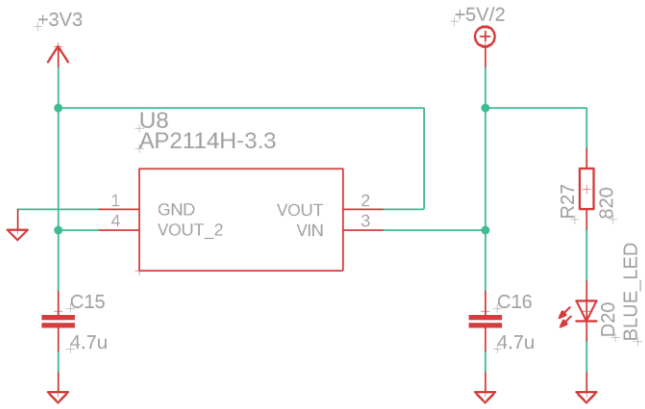


Figura 22: Regulador lineal 5V a 3V3 (esquemático Módulo de Relés)

- **Microcontrolador**

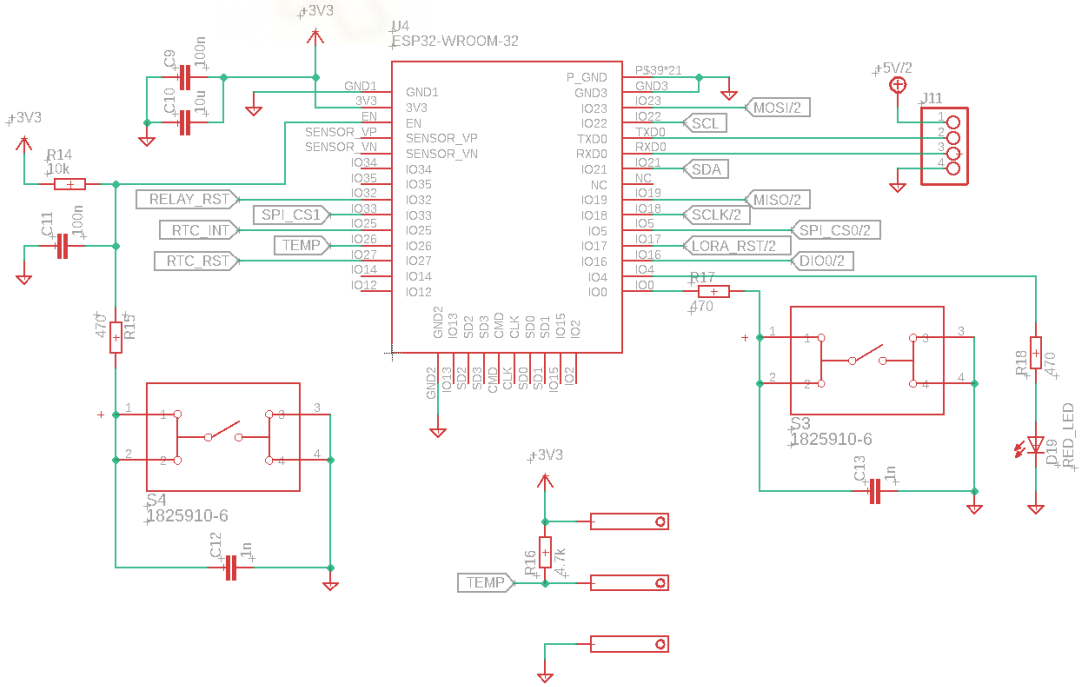


Figura 23: Microcontrolador y periféricos (esquemático Módulo de Relés)

- **Módulo LoRa**

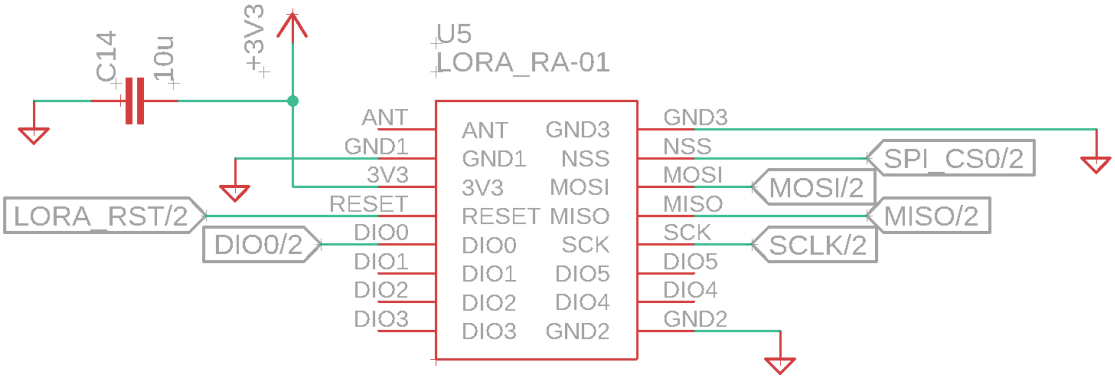


Figura 24: Módulo de radiofrecuencia LoRa (esquemático Módulo de Relés)

- **RTC**

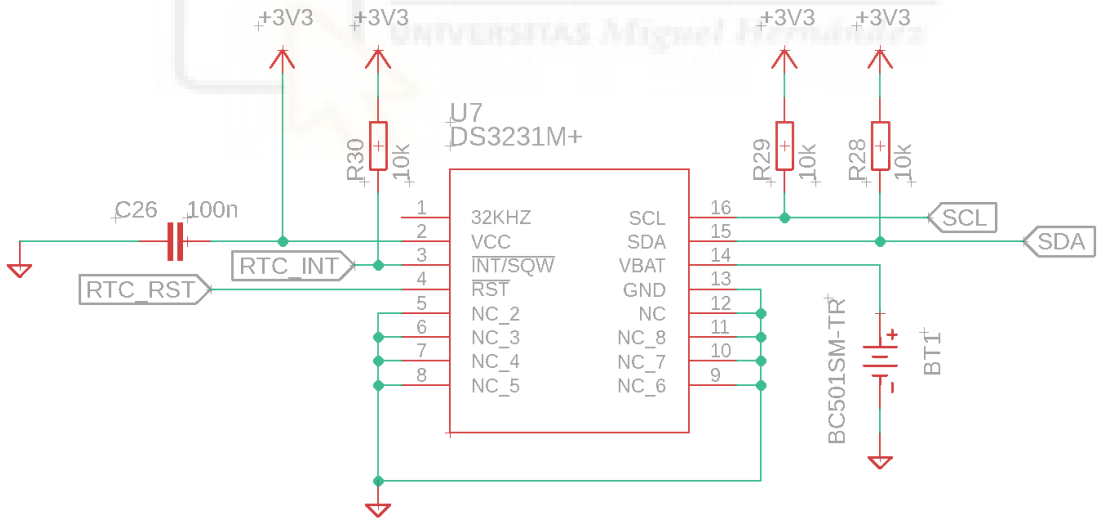


Figura 25: RTC (DS3231M, esquemático)

- **Relés**

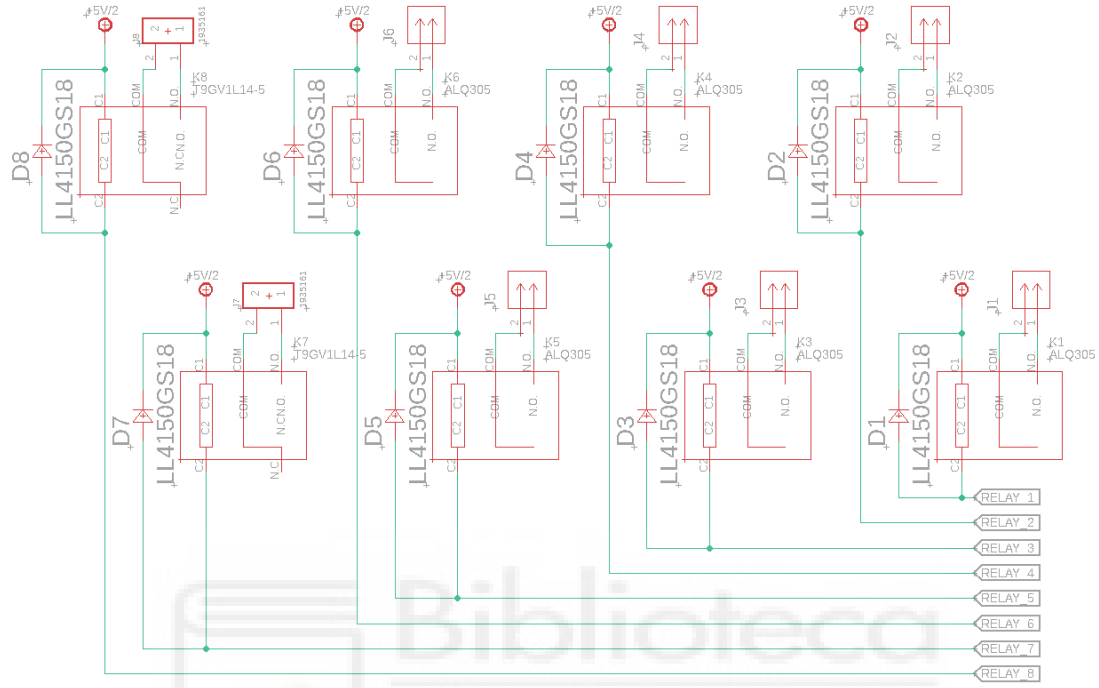


Figura 26: Relés (esquemático)

- **Driver de relés**

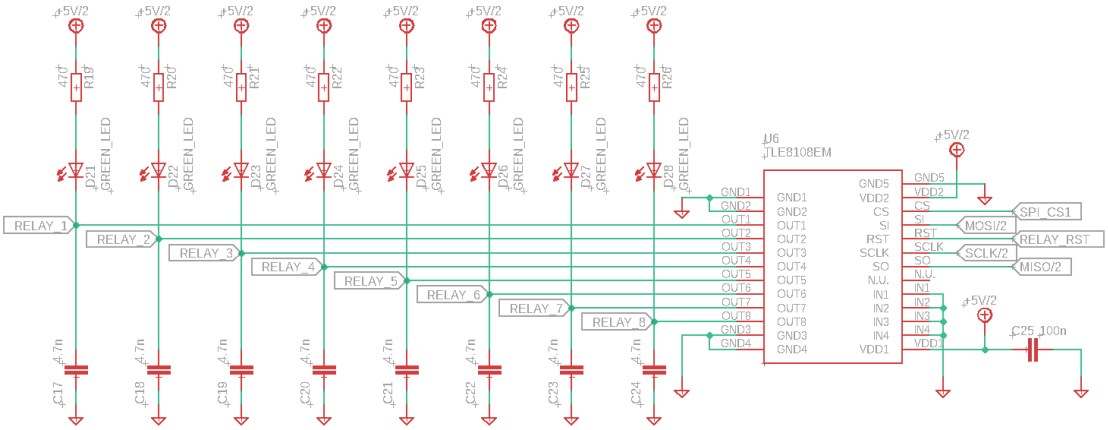


Figura 27: Driver de relés (TLE 8108 EM, esquemático)

3.2. JUSTIFICACIÓN DE LAS SOLUCIONES ADOPTADAS

3.2.1. MICROCONTROLADOR

Para poder controlar tanto las salidas como los led, o la comunicación entre los propios integrados necesitamos de un microcontrolador que sirva como maestro a los demás componentes.

Hay muchos microcontroladores en el mercado y muchos de ellos en la actualidad han alcanzado precios muy asequibles a pesar de poseer una gran capacidad de procesamiento.

Dentro de los microprocesadores disponibles en el mercado, para este producto se requiere que este sea capaz de conectarse a internet y actúe como servidor de una página web. Este requerimiento limita de manera significativa los microprocesadores disponibles. Tomaremos en consideración dos que cumplen todos los requerimientos:

- **ESP8266:**

  Es un chip de bajo costo desarrollado por la marca Espressif [15]. Lleva varios años en el mercado y ha demostrado ser una gran opción para el desarrollo de proyectos IoT. Cuenta con las siguientes características técnicas [16], [17]:

  - 32-bit RISC CPU a 80 MHz
  - 64 KiB de RAM de instrucciones y 96 KiB RAM de datos
  - QSPI flash externa de 512 KiB a 4 MiB (soportado hasta 16 MiB)
  - IEEE 802.11 b/g/n Wi-Fi
  - 16 GPIO pines
  - SPI, I^2C

  De estos 16 pines GPIO hay algunos que tienen una funcionalidad específica y no se podrán utilizar para este producto es por ello que requerimos de un microcontrolador más potente y con más salidas GPIO. Además el uso

continuado del wifi en este módulo aumenta considerablemente la temperatura del mismo.

- **ESP32:**

  Es un chip de bajo costo desarrollado por la marca Espressif [10]. Es una buena solución para el desarrollo de proyectos IoT que requieran una gran cantidad de recursos (alta frecuencia de reloj, gran cantidad de pines GPIO, etc.). Cuenta con las siguientes características técnicas [18], [19]:

  - Frecuencia de reloj de entre 160 MHz y 240 MHz
  - 520 KiB de RAM
  - Antena wifi integrada
  - Bluetooth 4,2 a 2,4 GHz; BT 2,0 y 4,0 BLE
  - 36 GPIO pines

  Gracias a su frecuencia de reloj y a su gran cantidad de pines de propósito general es ideal para nuestro producto.

Solución propuesta

Debido a la necesidad de una gran cantidad de pines GPIO para nuestro producto y un requerimiento de potencia bastante elevado debido al mantenimiento de un servidor, hemos optado por el chip ESP32, concretamente de su módulo ESP32-WROOM-32 [10].

3.2.2. CONEXIÓN ENTRE MÓDULOS

Como se desea poder controlar el PLC desde la vivienda, o en definitiva de un lugar alejado de la zona que se desea automatizar, es necesario que el PLC esté dividido en dos módulos y por lo tanto se requiere de una conexión entre estos para permitir la transferencia de información. Existen varias opciones a tener en cuenta:

- **Conexión a través de cable:**

El conexionado directo a través de cable entre los módulos asegura que la conexión sea fiable en todo momento, además de permitir el suministro de potencia (menos de 3W por hilo a 5V para RJ45). Uno de sus inconvenientes es el costo que supone para el consumidor el tener que llevar el cable desde el interior de la vivienda hasta el lugar donde se desee instalar el módulo exterior.

El disponer de una conexión alternativa a través de cable puede resultar conveniente para aquellas situaciones en las que el módulo UHF (*Ultra High Frequency*, frecuencia ultra alta) dé problemas, ya sea debido a falta de alcance de la señal, ambientes especialmente ruidosos o incluso al deterioro del propio módulo.

De todas formas descartaremos la opción de permitir la conexión a través de cable, ya que existen otros PLCs en el mercado con esta capacidad, además de que aumentaría significativamente el coste por parte del consumidor al tener que realizar la instalación del cableado.

- **Conexión mediante radiofrecuencia:**

La conexión radio, ya sea wifi, Bluetooth o UHF elimina tener que cablear ambos módulos y por lo tanto facilita su instalación, lo que hará más atractivo al producto. Su principal inconveniente es la dificultad de conseguir una conexión fiable y estable, ya que los módulos deben garantizar una buena conexión a una distancia 100 m habiendo obstáculos entre ellos, y de 500 m sin obstáculos. Es importante que se garanticen los 100 m con obstáculos, ya que entre ambos módulos es muy probable que haya varios tabiques e incluso que el módulo exterior se encuentre en una sala de máquinas bajo tierra.

Hay que tener en cuenta el posible retraso que haya desde que se envíe la señal hasta que el otro módulo la reciba, pero al no ser crítico el tiempo de respuesta en las actividades en la que está destinado el producto, no supondrá ningún inconveniente.

Para asegurar una buena calidad de conexión entre los módulos hay básicamente tres posibilidades:

- Aumentar la capacidad de recepción mediante amplificadores de bajo ruido para mejorar la calidad de la señal recibida.
- Disminuir la frecuencia de transmisión: disminuyendo la frecuencia a la que transmitimos la información nos permite obtener un mayor alcance a costa de disminuir el ancho de banda y de aumentar el tamaño de las antenas a utilizar.

De entre estas tres opciones utilizaremos la de disminuir la frecuencia de transmisión ya que es muy difícil aumentar el alcance de conexiones wifi o Bluetooth aunque se mejore significativamente su recepción. Además la tecnología wifi y Bluetooth está pensada para el envío de grandes cantidad de datos y para este producto necesitamos transmitir únicamente 85 bytes cada vez.

Por lo tanto descartaremos el uso de wifi y de Bluetooth, ya que su rango efectivo de funcionamiento no supera 50 metros en condiciones óptimas y mucho menos con obstáculos.

Solución propuesta

La conexión entre los módulos se hará mediante radiofrecuencia con módulos LoRa. El módulo en concreto será el LoRa Ra-01 el cual transmite a 433 MHz y según sus especificaciones es capaz de transmitir a hasta 10 kilómetros, aunque en las condiciones de obstáculos planteada (tabiques e incluso estando el receptor en una habitación subterránea) el rango efectivo será menor de 500m. Además este módulo es bajo: su precio de mercado ronda los 3 € [20].

Se ha decidido implementar únicamente la conexión mediante radiofrecuencia ya que se ha comprobado que garantiza el correcto funcionamiento de los módulos (ver especificaciones técnicas y campaña de pruebas para más información).

3.2.3. CONEXIÓN AL RÚTER

La conexión entre el Módulo Servidor y el rúter se puede realizar básicamente de dos formas:

- **Cableada (RJ45):**

Al igual que sucede con el punto anterior, la conexión mediante cable es la más fiable y asegura la conexión en todo momento entre el Módulo Servidor y el rúter.

Esta solución incrementaría el coste final y dificultad de instalación del producto al requerir de una clavija de conexión.

- **Mediante wifi:**

La conexión mediante wifi es muy cómoda al no requerir de cables, lo que facilita la instalación por parte del usuario. Además el Módulo Servidor es de dimensiones reducidas y está pensado simplemente para crear un servidor en la red wifi y enviar y recibir datos mediante los módulos LoRa, por lo que puede situarse cerca del rúter.

Por otra parte el microcontrolador [10] que se va a emplear para este producto ya incluye una antena wifi en la propia PCB, lo que reduce costes y facilita la instalación.

Solución propuesta

La conexión entre el Módulo Servidor y el rúter será mediante wifi. Utilizaremos la propia antena que viene incluida en la PCB del microcontrolador ESP32-WROOM-32 para su conexión.

3.2.4. SELECCIÓN DE DRIVER DE RELÉS

Para el accionamiento de los relés hay varias soluciones posibles. Debido a la limitada cantidad de corriente que puede suministrar las salidas de un microcontrolador, estas únicamente se pueden utilizar para alimentar leds o cargas menores de 20 mA (máxima corriente de salida de los GPIO en el ESP32). Es por ello que se necesita otra forma de poder alimentar los relés:

- **Mediante transistores discretos**

  Únicamente hay que conectar la salida del microcontrolador a la base del transistor, de tal forma que funcione como un interruptor entre la alimentación y la bobina del relé.

  Hay varias razones por las que no se utilizará este método. La primera es que se requiere un transistor para cada salida de relé, ocho en nuestro caso, y eso aumentaría la cantidad de rutado de la placa y posiblemente su tamaño.

  Por otra parte, conseguir que los transistores conmuten correctamente a 3,3V es bastante difícil ya que tendríamos una resistencia entre drenador y fuente considerable, lo que implica pérdidas debido a la disipación de calor en los transistores.

- **Mediante un driver específico**

  Los drivers específicos cumplen la tarea a la perfección además de incorporar funcionalidades adicionales como el diagnóstico del dispositivo para garantizar su correcto funcionamiento. Al estar los transistores incorporados dentro de un mismo IC, la resistencia que ofrece el transistor en conducción es extremadamente baja, lo que disminuye las pérdidas de encendido.

  Además el control de estos dispositivos se suele realizar mediante protocolos específicos, como el SPI, lo que disminuye la cantidad de señales a rutar.

Solución propuesta

Para controlar los relés se ha decidido utilizar el IC TLE8108EM [8] de la marca Infineon, el cual se alimenta a 5VDC (mismo voltaje que la bobina de los relés) y cuenta con una resistencia en conducción muy reducida: 1,7 Ω como máximo, a 150 ºC (Tj), resistencia típica de 0,8 Ω.

Para este producto se usarán los pines de entrada (IN1 – IN4) y por lo tanto, siguiendo las indicaciones de la ficha técnica, estos se conectarán a GND [8].

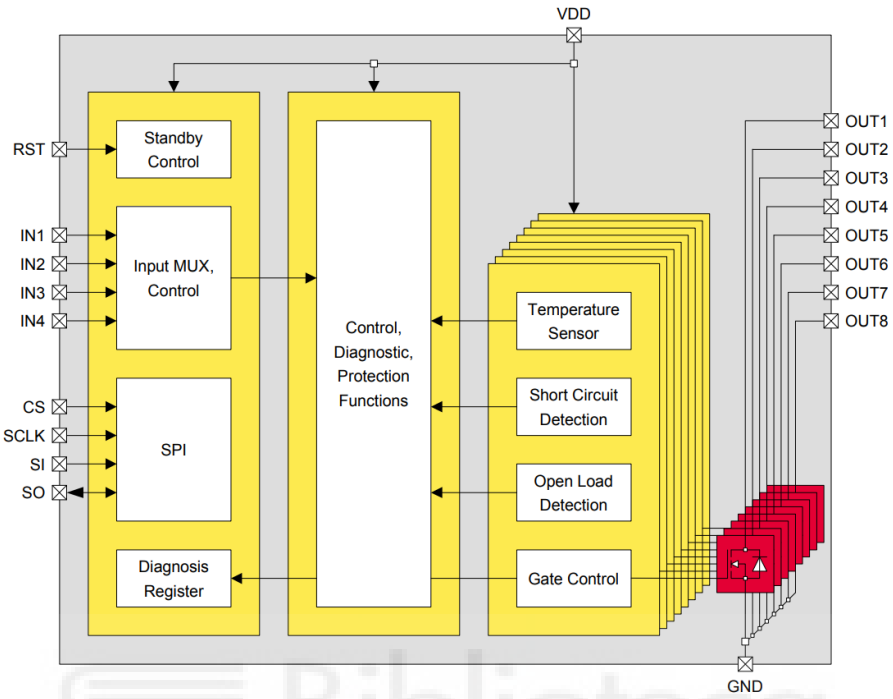


Figura 28: Diagrama de bloques del TLE8108EM [8, figure 1]

A pesar de que el TLE8108EM ya tiene un circuito de supresión de picos de voltaje producidos por la desconexión de la bobina del relé, se ha decidido colocar un diodo Flyback para cada salida, de forma que el integrado quede más protegido frente a picos de tensión producidos por la bobina del relé y aumente de esta forma la vida útil del integrado.

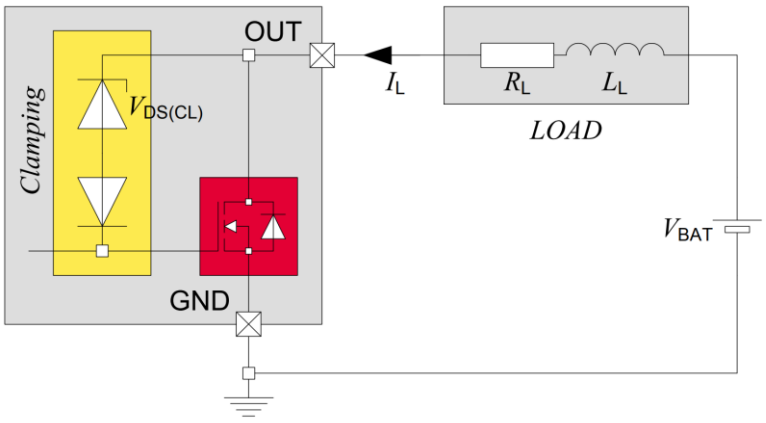


Figura 29: Diagrama del principio de funcionamiento del flyback interno del TLE8108EM [8, figure 6]

Además este integrado cuenta con funcionalidades de diagnóstico tales como cortocircuito a la fuente de alimentación (SCB, *Short circuit to battery*), cortocircuito a GND (SCG, *Short circuit to ground*), carga abierta (OL, *Open load*) y temperatura demasiado elevada (DOT, *Diagnosis of Overtemperature*).

3.2.5. ALIMENTACIÓN DE LOS MÓDULOS

La alimentación de ambos módulos será de 5VDC a través un conector micro USB, de forma que estos se puedan alimentar con cargadores de móvil y otras fuentes de alimentación que funcionen a este voltaje. De esta manera no hará falta adquirir un cargador específico para los módulos. Además la naturaleza de estos cargadores (normalmente aislados) aumenta la protección eléctrica aguas abajo.

La fuente de alimentación del Módulo Servidor tendrá que ser capaz de suministrar 300 mA (1.5 W), mientras que la del Módulo de Relés tendrá que ser capaz de suministrar 1 A (5 W).

La fuente de alimentación de los módulos deberá cumplir con la normativa UNE-EN 61558-1:2007 (Seguridad de los transformadores de potencia, fuentes de alimentación, bobinas de inductancia y productos análogos).

3.2.6. MÓDULOS UNIDOS EN LA PCB

Para reducir gastos en la producción he decidido incluir en el diseño de la PCB ambos módulos, tanto el Servidor como el de Relés. De esta forma se simplifica la fabricación de las PCBs y su montaje. Esto ha sido posible debido a la distribución de los componentes en módulos de 40x80 mm y 80x80 mm que conjuntamente compone una PCB de 120x80 mm.

Para facilitar su posterior separación se ha incluido un fresado longitudinal que divide los módulos (a excepción de los extremos).

3.3. LISTADO DE COMPONENTES Y PRESUPUESTO

El precio de fabricación de la PCB según el fabricante JLCPCB es de 5,80 € para 5 PCBs, de 44,06 € para 100 PCBs y de 378,91 € para 1000 PCBs [21].

El coste de las horas de ingeniero de diseño, tanto del esquemático como de la PCB no se tomarán en cuenta, ya que forman parte del desarrollo de este TFG.

El coste del software EAGLE, utilizado para el desarrollo de este producto, es de 0 € ya que existe un convenio entre la universidad Miguel Hernández y Autodesk. El precio de la licencia en caso de no existir este convenio es de 61 € mensuales. Se podría haber utilizado otros software como KiCad que son gratuitos.

3.3.1. Módulo SERVIDOR

Ref. Fabricante	Fabricante	Descripción	Encapsulado	Valor	Ref. Proveedor	Cantidad	Precio Unitario	Total	Total (100 mod)	Total (1000 mod)
ESP32-WROOM-32	Espressif Systems	Microcontrolador	SMD-38		1904-1010-1-ND	1	3,18 €	3,18 €	318,00 €	3.180,00 €
LoRa_Ra-01	Ai-Thinker Co., Ltd	LoRa Module (Not in Digikey)	SMD-16		-	1	3,00 €	3,00 €	300,00 €	3.000,00 €
AP2114H-3.3TRG1	Diodes Incorporated	3.3V Regulator 1A	SOT-223		AP2114H-3.3TRG1DICT-ND	1	0,29 €	0,29 €	14,85 €	84,49 €
1050171001	Molex	Micro USB connector			WM11262CT-ND	1	1,22 €	1,22 €	95,05 €	678,95 €
1825910-6	TE Connectivity ALCOSWITCH Switches	Push Button	TH		450-1650-ND	2	0,08 €	0,16 €	14,58 €	104,98 €
61300411121	Würth Elektronik	CONN HEADER VERT 4POS			732-5317-ND	1	0,15 €	0,15 €	8,04 €	73,74 €
150080RS75000	Würth Elektronik	LED RED CLEAR 0805 SMD	0805		732-4984-1-ND	1	0,15 €	0,15 €	11,90 €	103,06 €
150080VS75000	Würth Elektronik	LED GREEN CLEAR 0805 SMD	0805		732-4986-1-ND	8	0,15 €	1,20 €	82,45 €	824,48 €
150080BS75000	Würth Elektronik	LED BLUE CLEAR 0805 SMD	0805		732-4982-1-ND	1	0,15 €	0,15 €	11,90 €	103,06 €
CL21A106KOQNNNG	Samsung Electro-Mechanics	CAP CER 10UF 16V X5R 0805	0805	10u	1276-6455-1-ND	2	0,09 €	0,18 €	6,02 €	34,90 €
CL21A475KPFNNNE	Samsung Electro-Mechanics	CAP CER 4.7UF 10V X5R 0805	0805	4.7u	1276-1259-1-ND	2	0,08 €	0,16 €	5,72 €	32,10 €

CL21F104ZBCNNNC	Samsung Electro-Mechanics	CAP CER 0.1UF 50V Y5V 0805	0805	100n	1276-1007-1-ND	2	0,08 €	0,16 €	2,86 €	16,08 €
CL21B102KBANNNC	Samsung Electro-Mechanics	CAP CER 1000PF 50V X7R 0805	0805	1n	1276-1020-1-ND	2	0,08 €	0,16 €	3,16 €	17,64 €
CRCW080510K0FKEA	Vishay Dale	RES SMD 10K OHM 1% 1/8W 0805	0805	10k	541-10.0KCCT-ND	1	0,08 €	0,08 €	1,94 €	8,71 €
CRCW0805470RFKEAC	Vishay Dale	RES SMD 470 OHM 1% 1/8W 0805	0805	470	541-4147-1-ND	3	0,08 €	0,24 €	5,55 €	24,99 €
CRCW0805820RFKEA	Vishay Dale	RES SMD 820 OHM 1% 1/8W 0805	0805	820	541-820CCT-ND	1	0,08 €	0,08 €	1,94 €	8,71 €
CRCW0805270RFKEA	Vishay Dale	RES SMD 270 OHM 1% 1/8W 0805	0805	270	541-270CCT-ND	8	0,08 €	0,64 €	15,52 €	69,68 €

Total (1 mod)		Total (100 mod)		Total (1000 mod)	
11,20 €	Sin IVA	899,48 €	Sin IVA	8.365,57 €	Sin IVA
13,55 €	**Con IVA**	**1.088,37 €**	**Con IVA**	**10.122,34 €**	**Con IVA**

3.3.2. MÓDULO DE RELÉS

Ref. Fabricante	Fabricante	Descripción	Encapsulado	Valor	Ref. Proveedor	Cantidad	Precio Unitario	Total	Total (100 mod)	Total (1000 mod)
T9GV1L14-5	TE Connectivity Potter & Brumfield Relays	Relay 30A 5V	TH		PB2362-ND	2	2,93 €	5,86 €	440,40 €	3.425,34 €
ALQ305	Panasonic Electric Works	Relay 10A 5V	TH		255-3559-ND	6	0,89 €	5,34 €	331,81 €	2.986,26 €
TLE8108EMXUMA1	Infineon Technologies	Relay Driver 8 Channels	PG-SSOP-24		TLE8108EMXUMA1CT-ND	1	3,01 €	3,01 €	210,09 €	1.510,36 €
ESP32-WROOM-32	Espressif Systems	Microcontrolador	SMD-38		1904-1010-1-ND	1	3,18 €	3,18 €	318,00 €	3.180,00 €
LoRa_Ra-01	Ai-Thinker Co., Ltd	LoRa Module (Not in Digikey)	SMD-16		-	1	3,00 €	3,00 €	300,00 €	3.000,00 €
AP2114H-3.3TRG1	Diodes Incorporated	3.3V Regulator 1A	SOT-223		AP2114H-3.3TRG1DICT-ND	1	0,29 €	0,29 €	14,85 €	84,49 €
LL4150GS18	Vishay Semiconductor Diodes Division	Flyback Diode	SOD-80		LL4150GS18GICT-ND	8	0,13 €	1,04 €	25,81 €	165,92 €
1935161	Phoenix Contact	Wire to board 18A	TH		277-1667-ND	2	0,36 €	0,72 €	64,02 €	508,60 €

1984617	Phoenix Contact	Wire to board 10A	TH		277-1721-ND	6	0,40 €	2,40 €	184,55 €	1.705,02 €
1990012	Phoenix Contact	Wire to board 2A	TH		277-1795-ND	1	0,37 €	0,37 €	32,95 €	261,74 €
1050171001	Molex	Micro USB connector			WM11262CT-ND	1	1,22 €	1,22 €	95,05 €	678,95 €
1825910-6	TE Connectivity ALCOSWITCH Switches	Push Button	TH		450-1650-ND	2	0,08 €	0,16 €	14,58 €	104,98 €
DS3231M+TRL	Maxim Integrated	RTC	SOIC-16		DS3231M+TRLCT-ND	1	6,21 €	6,21 €	464,34 €	4.043,38 €
BC501SM	MPD (Memory Protection Devices)	Coin Cell Support	SMD	1220	BC501SM-ND	1	0,86 €	0,86 €	60,35 €	487,93 €
61300411121	Würth Elektronik	CONN HEADER VERT 4POS			732-5317-ND	1	0,15 €	0,15 €	8,04 €	73,74 €
150080RS75000	Würth Elektronik	LED RED CLEAR 0805 SMD	0805		732-4984-1-ND	1	0,15 €	0,15 €	11,90 €	103,06 €
150080VS75000	Würth Elektronik	LED GREEN CLEAR 0805 SMD	0805		732-4986-1-ND	8	0,15 €	1,20 €	82,45 €	824,48 €
150080BS75000	Würth Elektronik	LED BLUE CLEAR 0805 SMD	0805		732-4982-1-ND	1	0,15 €	0,15 €	11,90 €	103,06 €
CL21A106KOQNNNG	Samsung Electro-Mechanics	CAP CER 10UF 16V X5R 0805	0805	10u	1276-6455-1-ND	2	0,09 €	0,18 €	6,02 €	34,90 €
CL21A475KPFNNNE	Samsung Electro-Mechanics	CAP CER 4.7UF 10V X5R 0805	0805	4.7u	1276-1259-1-ND	2	0,08 €	0,16 €	5,72 €	32,10 €
CL21F104ZBCNNNC	Samsung Electro-Mechanics	CAP CER 0.1UF 50V Y5V 0805	0805	100n	1276-1007-1-ND	4	0,08 €	0,32 €	5,72 €	32,16 €
CL21B472KCANNNC	Samsung Electro-Mechanics	CAP CER 4700PF 100V X7R 0805	0805	4.7n	1276-2524-1-ND	8	0,08 €	0,64 €	8,80 €	69,20 €
CL21B102KBANNNC	Samsung Electro-Mechanics	CAP CER 1000PF 50V X7R 0805	0805	1n	1276-1020-1-ND	2	0,08 €	0,16 €	3,16 €	17,64 €
CRCW080510K0FKEA	Vishay Dale	RES SMD 10K OHM 1% 1/8W 0805	0805	10k	541-10.0KCCT-ND	4	0,08 €	0,32 €	7,76 €	34,84 €
CRCW08054K70FKEAC	Vishay Dale	RES SMD 4.7K OHM 1% 1/8W 0805	0805	4.7k	541-4131-1-ND	1	0,08 €	0,08 €	1,85 €	8,33 €
CRCW0805470RFKEAC	Vishay Dale	RES SMD 470 OHM 1% 1/8W 0805	0805	470	541-4147-1-ND	11	0,08 €	0,50 €	9,16 €	91,63 €
CRCW0805820RFKEA	Vishay Dale	RES SMD 820 OHM 1% 1/8W 0805	0805	820	541-820CCT-ND	1	0,08 €	0,08 €	1,94 €	8,71 €

Total (1 mod)		Total (100 mod)		Total (1000 mod)	
37,75 €	Sin IVA	2.721,22 €	Sin IVA	23.576,82 €	Sin IVA
45,67 €	**Con IVA**	**3.292,67 €**	**Con IVA**	**28.527,95 €**	**Con IVA**

3.4. PCB

La PCB (*printed circuit board*, placa de circuito impreso) ha sido diseñada con el CAD EAGLE, utilizando un tamaño de placa de 120x80 mm, de tal forma no haya que mandar a fabricar una placa para cada módulo, sino que la PCB incluya ambos módulos.

Esta ha sido fabricada por la empresa JLCPCB, con un grosor de cobre de 0,035 mm.

Los planos de ambos módulos los podemos encontrar en el Anexo II.

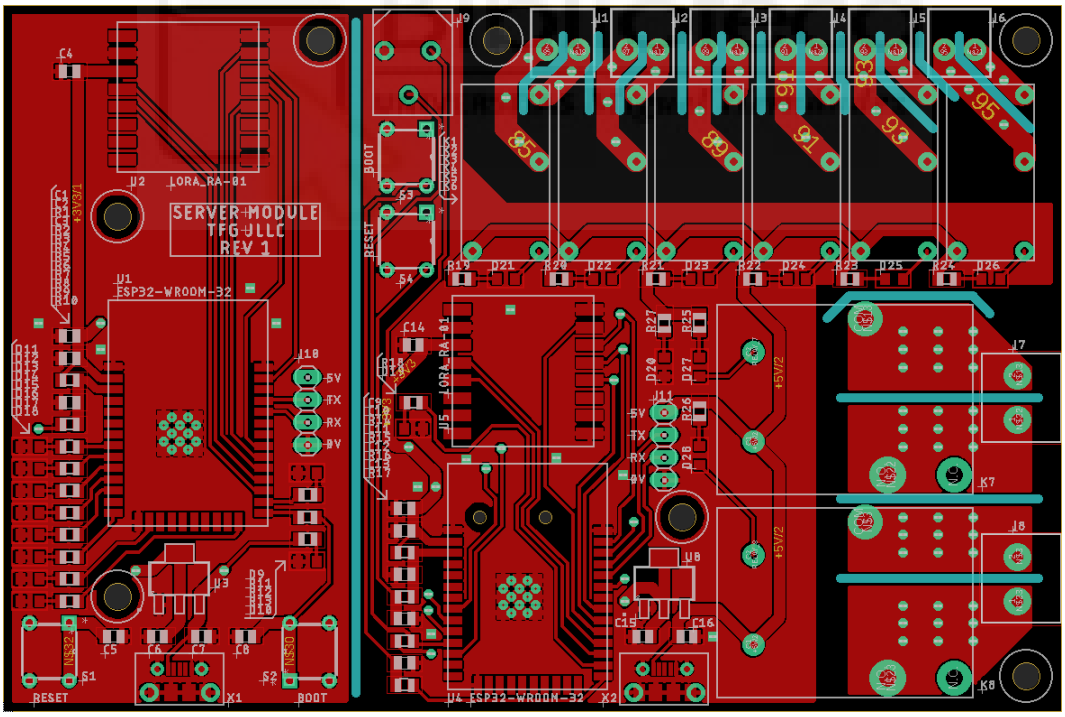


Figura 30: Modelo CAD de la PCB (cara top)

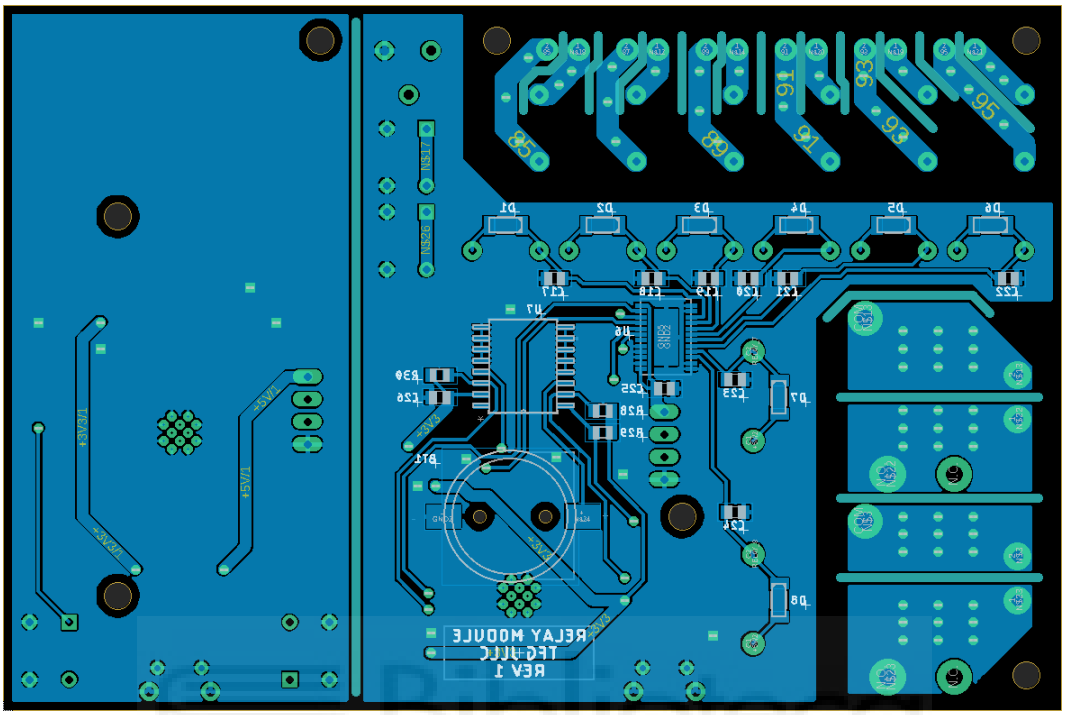


Figura 31: Modelo CAD de la PCB (cara bottom)

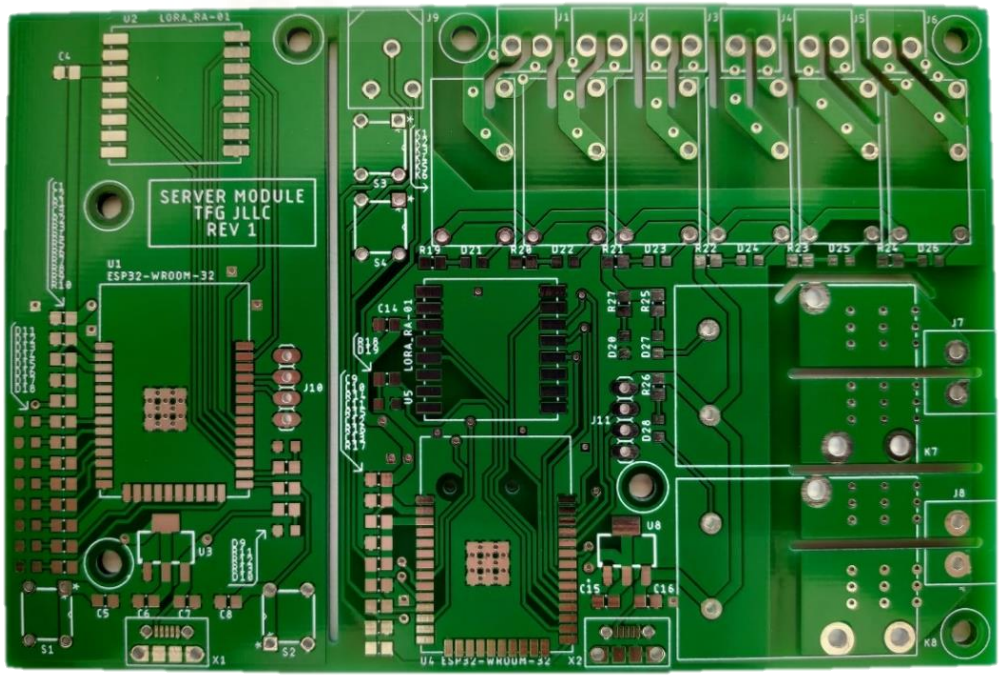


Figura 32: PCB fabricada

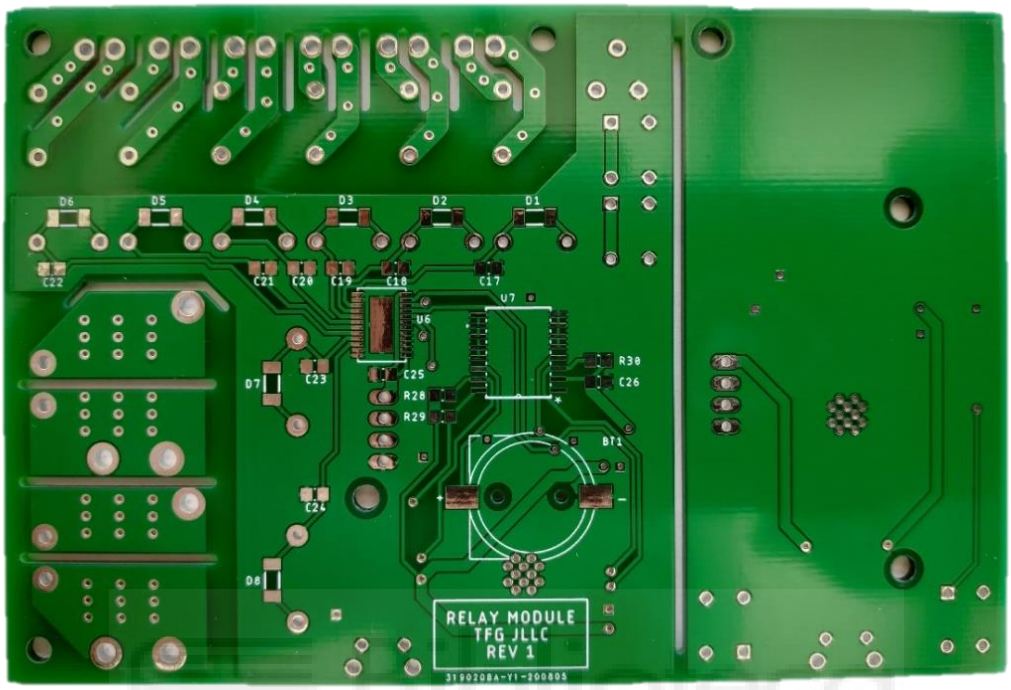


Figura 33: PCB fabricada (reverso)

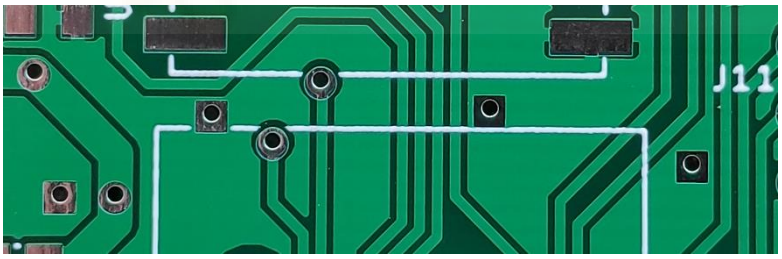


Figura 34: Detalle VIAS

3.4.1. CONSIDERACIONES ADICIONALES AL DISEÑO

Debido a la alta intensidad que puede llegar a darse en las salidas de los relés (hasta 17,5A @250V en los relés para motores y hasta 5A @250V en los relés genéricos), se ha decidido rutar la señal tanto por la cara superior como por la inferior en la señal que une los relés con las salidas y además unir la señal con varias VIAS. En el caso de los relés

para motores se ha puesto la señal como un plano y una matriz de VIAS que unir ambos planos.

Ya que las salidas están diseñadas para trabajar normalmente a 250VAC o incluso con más en el caso de las salidas 7 y 8, con tal de aumentar la seguridad y disminuir el riesgo de arco eléctrico, se ha decidido fresar las zonas de la PCB que separan las salidas y entre ambos pines de cada propia salida.

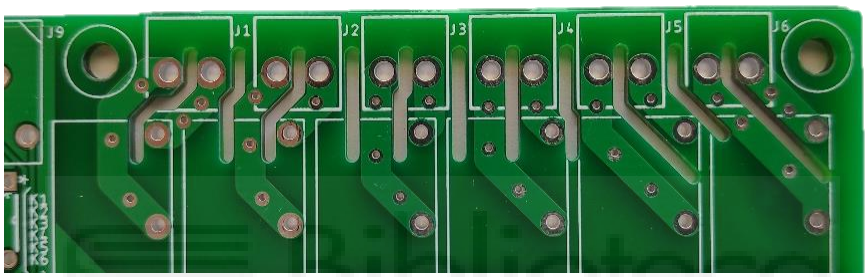


Figura 35: Detalle del fresado y VIAS en las salidas (1 de 2)

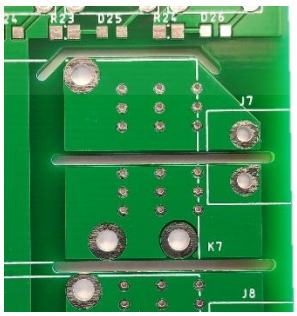


Figura 36: Detalle del fresado y VIAS en las salidas (2 de 2)

4. SOFTWARE

El código para la programación del microcontrolador ESP32 WROOM 32 ha sido escrito en Arduino, ya que Espressif, fabricante del chip ESP32, añadió la posibilidad de compilar el código para el microcontrolador desde el IDE de Arduino, lo que facilita la programación de este microcontrolador, ya que podemos utilizar muchas de las librerías y funciones de Arduino.

Los programas de las salidas se almacenan en la EEPROM del ESP32 de forma que aunque el módulo se desconecte de la alimentación los programas quedarán almacenados en el microcontrolador.

A modo de indicador el led rojo se encenderá mientras se esté transmitiendo datos.

El código necesario para la programación de ambos módulos se puede descargar desde el siguiente enlace: github.com/Kuenlun/PLC-ESPLoRa

4.1. ESTRUCTURA DEL MENSAJE ENVIADO A TRAVÉS DEL MÓDULO LORA

Los módulos de radiofrecuencia LoRa Ra-01 a pesar de ser extremadamente eficaces para el envío de datos a largas distancia [6] y de tener un bajo consumo, están limitados al envío de hasta 255 bytes cada vez. Por ello con tal de simplificar el envío de datos, el tamaño del mensaje que se transmita será menor de 255 bytes, concretamente de 85 bytes de forma que en cada transmisión se pueda enviar toda información necesaria para los módulos.

En el código, este mensaje se almacena en un array de bytes de 85 posiciones. Estos bytes están estructurados de la siguiente manera:

Índice del byte	Dato	Tipo de dato
0	Segundo	unsigned int8
1	Minuto	unsigned int8
2	Hora	unsigned int8
3	Día	unsigned int8
4	Mes	unsigned int8
5 - 6	Año	unsigned int16
7	Estados[1)]	byte
8	Selector[1)]	byte
9 - 10	Temperatura del RTC	Personalizado[2)]
11 - 12	Temperatura de la sonda	Personalizado[2)]
13 - 21	Programa de la salida 1	Personalizado[3)]
22 - 30	Programa de la salida 2	Personalizado[3)]
31 - 39	Programa de la salida 3	Personalizado[3)]
40 - 48	Programa de la salida 4	Personalizado[3)]
49 - 57	Programa de la salida 5	Personalizado[3)]
58 - 66	Programa de la salida 6	Personalizado[3)]
67 - 75	Programa de la salida 7	Personalizado[3)]
76 - 84	Programa de la salida 8	Personalizado[3)]

1) Un bit para cada salida (LSB, salida 1).
2) De MSB a LSB: 1 bit de signo, 11 bits para la parte entera, 4 bits para la parte decimal.
3) Bytes 0 – 5: programación horaria, siendo cada bit media hora (LSB del primer byte equivale a 00:00 – 00:30)
   Byte 6: programación semanal (LSB, domingo).
   Bytes 7 – 8: programación mensual (LSB, enero).

La variable “selector” determina si se empleará el programa almacenado en memoria o la variable “estados” para el encendido o apagado de cada salida (1 bit para cada salida). 0 equivale a utilizar la variable “estados” y 1 equivale a utilizar el programa.

La variable “estados” almacena si las salidas están encendidas o apagadas cuando el bit correspondiente a cada salida de la variable “selector” vale 0.

4.2. COMENTARIOS SOBRE EL CÓDIGO DEL MÓDULO SERVIDOR

El código del Módulo Servidor se basa en el proyecto "ESP32 Web Server – Arduino IDE" [22]. En este proyecto Rui Santos crea un servidor con el ESP32 para que con el smartphone podamos encender un led alimentado directamente desde un GPIO (*General Purpose Input/Output*, Entrada/Salida de Propósito General) del ESP32.

Este código ha sido modificado para crear una página web más compleja desde la cual se pueda encender, apagar y programar las ocho salidas de relé. Además desde esta página web pueden visualizarse diversas variables:

- Fecha y hora del IC RTC DS3231M
- Temperatura interna del IC RTC DS3231M
- Temperatura de la sonda de temperatura (sensor DS18B20)
- Calidad de la conexión LoRa (RSSI)

5. VALIDACIÓN EXPERIMENTAL

Debido a la situación producida por la pandemia del coronavirus no se ha podido acudir al laboratorio de la universidad para la realización de las pruebas necesarias.

Posibles test adicionales a realizar:

- Fluke Ti450: Esta cámara térmica nos ayuda a encontrar que componentes están disipando más energía en forma de calor. Debemos asegurar que todos los componentes que hemos seleccionado para los módulos son capaces de disipar el calor correctamente, prestando especial atención al microcontrolador ESP32, al módulo LoRa, al driver de los relés, y a las pistas y planos que van a soportar corrientes significativas, como las salidas de los relés.
- Emisiones electromagnéticas (EMC): Este test es especialmente relevante si se desea vender la placa, ya que hay que cumplir las normativas relativas a las emisiones electromagnéticas. Este test debe ser realizado en un laboratorio específico.

A pesar de las limitaciones por la situación del COVID-19, se ha podido verificar el correcto funcionamiento del sistema a través de las siguientes campañas de pruebas en entorno real:

5.1. FIABILIDAD DE LA CONEXIÓN LORA

Para comprobar la calidad de la conexión LoRa, aprovecharemos una función incluida en la librería de Arduino LoRa, elaborada por Sandeep Mistry [23]. Esta función es “packetRssi()”, la cual devuelve la calidad de la recepción de un mensaje enviado a través de los módulos LoRa.

En la página web principal desarrollada para este producto se puede observar fácilmente el valor devuelto por esta función. El valor de la calidad de la conexión que podemos observar en la página (RSSI) es la calidad de recepción del mensaje de contestación que

envía el Módulo de Relés al Módulo Servidor (para más información referirse al apartado 6.5.)

Para la realización de esta prueba una persona se situará con su dispositivo móvil conectado a la misma red wifi que el Módulo Servidor, pulsando el botón "Sincronizar" constantemente y apuntando la calidad de la conexión que se muestra en la página. Otra persona estará sujetando el Módulo de Relés junto con una batería portátil mientras se va alejando del Módulo Servidor.

5.1.1. RESULTADOS OBTENIDOS

Durante las pruebas realizadas en mi domicilio obtuve los siguientes resultados:

Distancia (m)	RSSI (dBm)
1	-37
5	-55
10	-69
25	-79
50	-84
100	-88
200	-95

Si hacemos un gráfico de RSSI frente a la distancia entre módulos con escala horizontal logarítmica nos damos cuenta de se aproxima bastante a una recta (R^2 = 0,976).

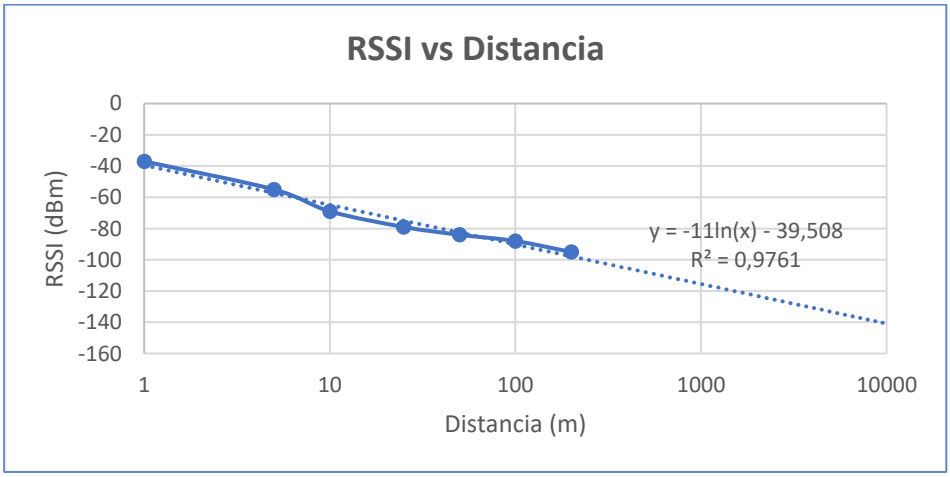


Figura 37: Gráfico de la calidad de la conexión LoRa (línea de tendencia)

Si lo que proclama la ficha técnica del módulo LoRa Ra-01 [11] es cierto, este sería capaz de alcanzar una sensibilidad de hasta -141 dBm. Extrapolando los datos obtenidos del ensayo, a -141 dBm obtendríamos un alcance de 10 km.

Este ensayo nos muestra que para los usos que está pensado el producto (automatización de bombas de piscina, de riego y electroválvulas), los cuales no suelen estar a más de 200 m, la conexión entre los módulos será suficiente.

5.2. FIABILIDAD DE LA CONEXIÓN WIFI

Para la realización de esta prueba conectaremos el Módulo Servidor a un ordenador portátil a través del puerto serie mediante el conversor USB a TTL UART y observaremos el puerto serie en el IDE de Arduino. Nos iremos alejando del rúter wifi mientras pulsamos el botón "RESET" del módulo. En el momento en el que el Módulo Servidor no pueda conectarse a la red wifi pararemos la prueba y apuntaremos la distancia al rúter.

Repetiremos la prueba en diferentes situaciones: sin obstáculos y con obstáculos (situación común, diversos tabiques entre el rúter y el Módulo Servidor).

5.2.1. RESULTADOS OBTENIDOS

El módulo ha logrado conectarse a la red wifi hasta una distancia de 20 m en línea recta y sin tabiques te por medio. Sin embargo, a esta distancia solo se conectaba algunas veces mientras que otras no conseguía conectarse. A 15 m sin obstáculos logró conectarse todas las veces.

Cuando había un tabique entre el módulo y el rúter la distancia máxima a la que todavía lograba conectarse era de 10 m pero la conexión no siempre estaba asegurada. A 5 m habiendo tabiques entre la conexión wifi y el módulo no hubo ningún problema de conexión.

5.3. CONSUMO ENERGÉTICO DE LOS MÓDULOS

Para la realización de esta prueba conectaremos los módulos a la alimentación con un amperímetro en serie de forma que podamos medir la corriente que estos demandan. Asumiremos una tensión de alimentación constante de 5V (DC). Primero realizaremos la prueba con el Módulo Servidor y luego con el Módulo de Relés.

La prueba la realizaremos en diferentes situaciones: cuando los módulos estén en reposo y cuando estén enviando datos. En el caso del Módulo de Relés obtendremos la demanda de corriente de cada uno de los relés por separado para observar su consumo.

El amperímetro (multímetro) empleado para esta prueba es un "Xindar DP1000.031" [24].

5.3.1. RESULTADOS OBTENIDOS

El consumo de los módulos en reposo es de 75 mA. Cuando los módulos envían datos a través de los módulos LoRa llegan a añadir un consumo adicional de hasta 100 mA de forma puntual.

- **Módulo Servidor**

Cada led indicador de las salidas añade un consumo de 4,4 mA cuanto se encuentra encendido.

$$A_S = 75 + 100T_x + 4{,}4Q$$

$Donde\ Q\ es\ el\ número\ de\ salidas\ encendidas\ y\ T_x = 1\ si\ se\ está\ enviando\ datos$

El consumo máximo de este módulo es de 110,2 mA y se alcanza estando todas las salidas encendidas. De forma puntual puede llegar hasta los 210,2 mA.

- **Módulo de Relés**

En el caso del Módulo de Relés cada relé genérico añade un consumo de 40 mA y cada relé de potencia un consumo de 180 mA.

$$A_R = 75 + 4{,}4(Q_G + Q_M) + 40Q_G + 180Q_M$$

$$Donde\ Q_G\ es\ el\ número\ de\ salidas\ genéricas\ encendidas\ y$$
$$Q_M\ es\ el\ número\ de\ salidas\ para\ motores\ encendidas$$

El consumo máximo de este módulo es de 710,2 mA y se alcanza estando todos los relés encendidos y por lo tanto $Q_G = 6$ y $Q_M = 2$. De forma puntual puede llegar hasta los 810,2 mA.

Dado que la alimentación de los módulos es a 5VDC para calcular la potencia consumida no tenemos más que multiplicar la intensidad por 5V.

6. MANUAL DE INSTRUCCIONES

6.1. PROGRAMAS Y LIBRERIAS NECESARIAS

Para la programación de los módulos es necesario que tengamos instalado el IDE (*Integrated Development Environment*, entorno de desarrollo integrado) de Arduino en un ordenador. Para ello debemos ir al siguiente enlace: "arduino.cc/en/main/software".

Una vez instalado Arduino se debe incluir algunas librerías necesarias para los módulos. Para ello abriremos el Arduino IDE e iremos al menú "Herramientas" y dentro de él haremos clic en "Administrar Bibliotecas…". En la ventana que aparecerá buscamos e instalamos las siguientes librerías:

Para enviar y recibir mensajes mediante los módulos LoRa

- LoRa by Sandeep Mistry [23]

Para poder utilizar el IC RTC DS3231

- RTClib by Adafruit [25]

Para leer la temperatura de la sonda (sensor DS18B20)

- OneWire by Jim Studt, Tom Pollard, Robin James, Paul Stoffregen, *et al.* [26]
- DallasTemperature by Miles Burton, Tim Newsome, Guil Barros, Rob Tillaart. [27]

También deberemos instalar una extensión de Arduino que permite compilar el código para el microcontrolador ESP32. En los siguientes enlaces encontraremos una guía de cómo hacerlo, tanto para Windows como para MacOS o Linux:

- Guía para Windows [28]: randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions
- Guía para Mac OS y Linux [29]: randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-mac-and-linux-instructions

Una vez tengamos instalado el IDE de Arduino y las librerías tendremos que descargar el código que utilizaremos para programar los módulos. Lo podemos descargar desde el siguiente enlace: github.com/Kuenlun/PLC-ESPLoRa

Antes de programar los módulos debemos modificar las siguientes líneas del archivo "Servidor.ino":

```
// Replace with your network credentials
const char* ssid = "ssid";
const char* password = "pass";
```

Debemos reemplazar el *string* "ssid" por el nombre de nuestra red wifi y el *string* "pass" por la contraseña. Una vez realizados los cambios guardamos el archivo.

6.2. PROGRAMACIÓN DE LOS MÓDULOS

A continuación procederemos a programar los módulos. Para ello necesitaremos un conversor USB a TTL UART. Lo conectaremos siguiendo el siguiente esquema al módulo servidor:

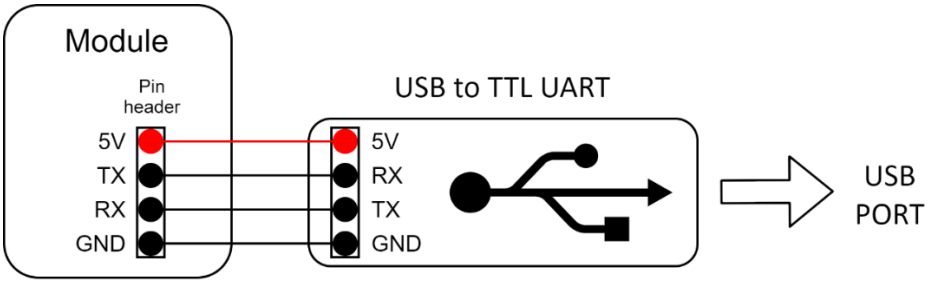


Figura 38: Diagrama de conexión del conversor USB a TTL UART

Una vez tengamos conectado el conversor abriremos el archivo "Servidor.ino".

Debemos configurar el dispositivo al que vamos a subir el código. Para ello iremos a la pestaña "Herramientas" y en "Placa" seleccionaremos "ESP32 Dev Module". Los demás parámetros los podemos dejar por defecto.

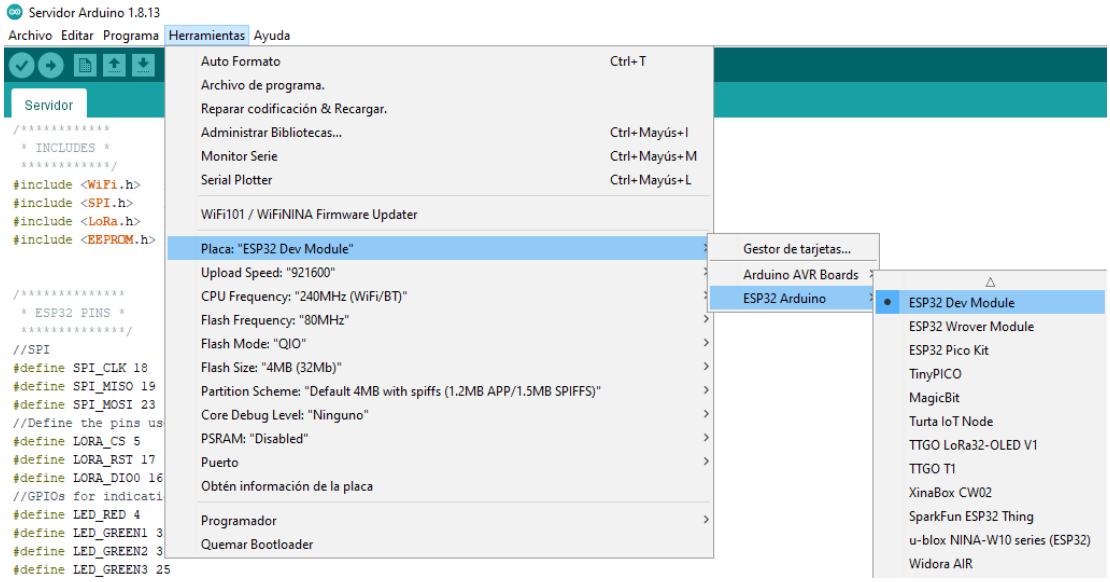


Figura 39: Selección de la placa en el IDE de Arduino

A continuación, seleccionaremos el puerto COM al que hemos conectado el USB y le daremos al botón “Subir”.

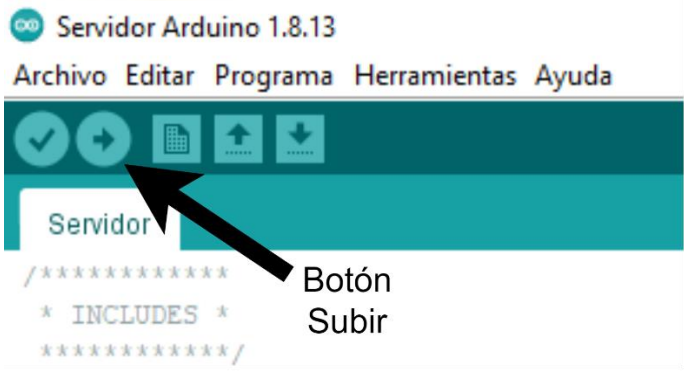


Figura 40: Botón "Subir" del IDE de Arduino

Cuando el código esté compilado y listo para subir a la placa deberemos presionar la siguiente combinación de botones en el módulo:

1. Presionar “BOOT” y “RESET” y mantenerlos presionados.

2. Dejar de presionar “RESET” mientras continuamos presionando “BOOT”.
3. Una vez veamos que el módulo se está programando ya podremos dejar de presionar “BOOT”.

Para la programación del módulo de relés debemos realizar el mismo procedimiento con el archivo “Reles.ino”.

Deberemos programarlo dos veces. Esto es debido a que el ajuste de la hora del reloj interno del módulo se realiza obteniendo la fecha y hora del ordenador desde el cual se programa. Este ajuste se realiza en la parte del código denominada “Setup”, la cual se ejecuta cada vez que el módulo se enciende. Si únicamente lo programamos con el código que ajusta la hora, cada vez que este se reinicie cambiará la fecha y hora a la del momento en que se programó.

Por lo tanto, la primera programación la haremos sin modificar el código, para que el RTC ajuste su reloj con la hora de nuestro ordenador y la segunda comentando la línea de código que ajusta la hora, ya que el RTC mantendrá la mantendrá.

Línea de código a comentar en “Reles.ino”:

```
// Update the RTC time
rtc.adjust(DateTime(__DATE__, __TIME__));
```

6.3. PUESTA EN MARCHA DEL MÓDULO SERVIDOR

Una vez tengamos los módulos programados procederemos a la instalación de los módulos, comenzando por el Módulo Servidor.

1. **Alimentación:**

Situar el módulo servidor cerca del rúter wifi y conectar a la alimentación mediante el cable micro USB. La luz del led azul indicará que el módulo tiene alimentación.

Se recomienda alimentar el módulo a través de una fuente de alimentación aislada.

2. **Obtención de la dirección IP:**

Una vez el módulo tenga alimentación, este actuará como servidor al cual nos podremos conectar.

Para acceder al servidor necesitamos estar conectados a la misma red wifi que el módulo y conocer la dirección IP en la que se encuentra. Para ello tenemos dos métodos:

- **Mediante el rúter**

  Debemos escribir en la barra de dirección de cualquier navegador la siguiente dirección: "192.168.0.1".

  Una vez dentro de la configuración del rúter, buscaremos la lista de IPs que este ha asignado a los dispositivos conectados y apuntaremos aquella que se corresponda con el Módulo Servidor.

- **Mediante el puerto serie (*pin header*)**

  Conectaremos el conversor USB a TTL al *pin header* del módulo y al puerto USB del ordenador y abriremos el IDE de Arduino. Dentro de Arduino seleccionaremos el puerto COM al que hemos conectado el USB y abriremos el monitor serie.

  Si reiniciamos el módulo presionando el botón "RESET" deberíamos poder ver en el monitor serie la dirección IP que el rúter ha asignado al módulo.

En caso de que cambie la IP asignada al módulo deberemos repetir el proceso de obtención de la IP.

3. **Acceso al servidor:**

Una vez tengamos la dirección IP del servidor la introduciremos en la barra de dirección del navegador. Se debería poder ver entonces la página web generada por el Módulo Servidor (ver figura 43).

6.4. PUESTA EN MARCHA DEL MÓDULO DE RELÉS

1. Cableado:

Cada salida del módulo corresponde con un relé y por lo tanto funciona como un interruptor entre los dos pines de cada salida. Todas las salidas son normalmente abiertas. Para su correcta conexión se recomienda seguir el siguiente esquema:

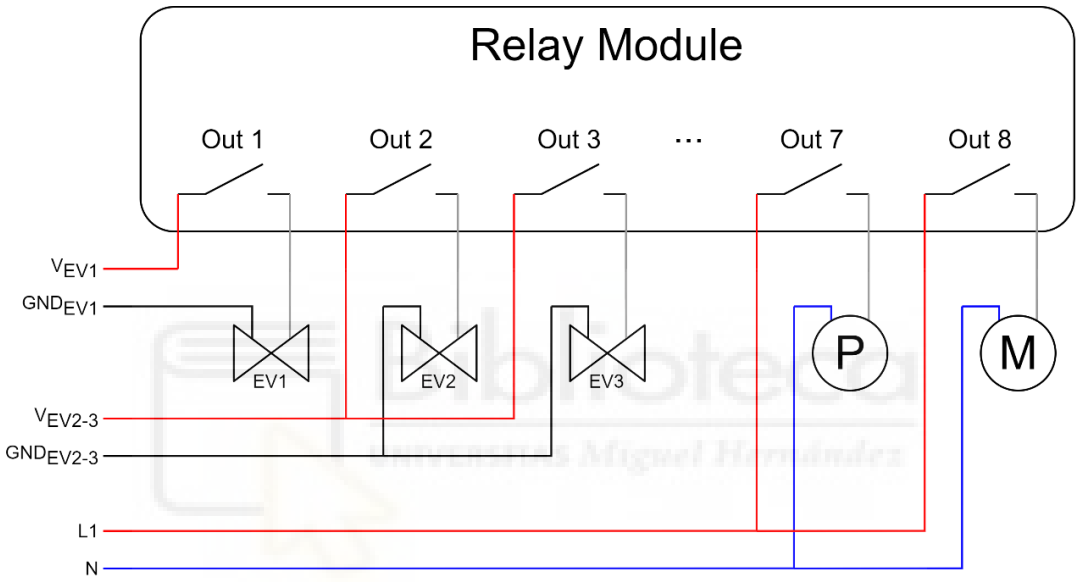


Figura 41: Diagrama de conexión de salidas del Módulo de Relés

Se debe conectar la fase a las salidas de los relés y no el neutro, de esta forma el dispositivo aguas abajo quedará protegido. El módulo cumple con la normativa UNE-EN 60335-1:2012/A13:2017 (Aparatos electrodomésticos y análogos. Seguridad) y UNE-EN 60730-1:2019 (Dispositivos de control eléctrico automáticos).

En caso de que se desee conectar la sonda de temperatura (sensor DS18B20) deberemos seguir el siguiente esquema:

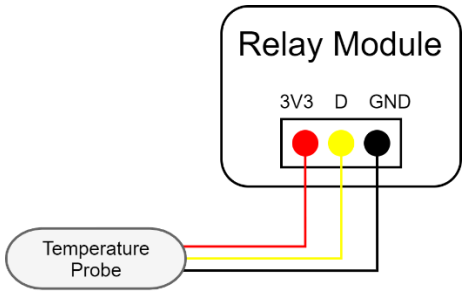


Figura 42: Diagrama de conexión de sonda de temperatura (DS18B20)

Debido a las condiciones de humedad en las que se encontrará el módulo, este deberá ir protegido dentro de una caja de conexiones estanca.

2. Alimentación:

Conectar a la alimentación mediante el cable micro USB. La luz del led azul indicará que el módulo tiene alimentación.

Se recomienda alimentar el módulo a través de una fuente de alimentación aislada.

6.5. CONTROL DEL PLC A TRAVÉS DE LA PÁGINA WEB

Desde la página web generada por el Módulo Servidor podremos encender, apagar y programar las ocho salidas de potencia además de visualizar algunos datos adicionales, como la fecha, hora y la temperatura del módulo de relés, así como la temperatura de la sonda para el agua de la piscina (opcional) y la calidad de conexión entre los módulos.

Es importante saber que los datos sobre la programación y estado de las salidas del Módulo Servidor prevalecen sobre los del Módulo de Relés. Si se encuentran discrepancias durante el envío de información entre los módulos, el Módulo de Relés actualizará los datos por los que recibe del Módulo Servidor.

La página web ha sido diseñada específicamente para facilitar su uso en dispositivos móviles, ajustando el contenido de la página a la pantalla desde la que se ve y

disponiendo los botones y los paneles de control en filas reducidas para que la página no tenga problemas de visualización en dispositivos con un ancho de pantalla mayor de 370 píxeles. Utiliza colores para los botones de forma que sea todavía más sencilla su comprensión.

Cuando accedamos a la página principal del servidor podremos observar lo siguiente:

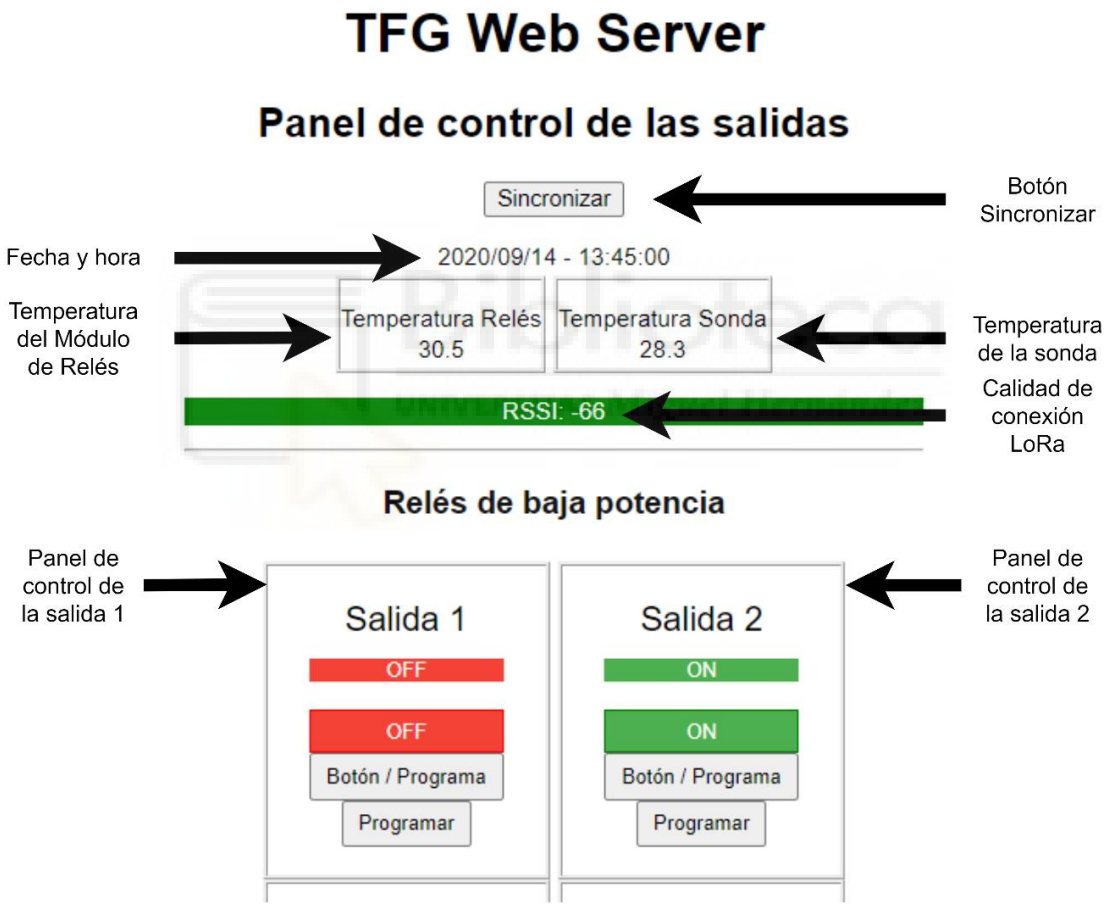


Figura 43: Página web principal creada por el Módulo Servidor

En la parte superior encontramos un botón con la etiqueta “Sincronizar”. Si lo pulsamos, el Módulo Servidor enviará un mensaje al Módulo de Relés con los programas y estados que tiene almacenado. El Módulo de Relés contestará enviando toda la información de

que disponga: fecha y hora, temperaturas, estado de las salidas, selector y programas. A partir de esta información la página web se actualizará y mostrará los nuevos datos.

Todos los demás botones de la página principal (a excepción de los botones con la etiqueta “Programar”) realizan la misma función que el de “Sincronizar” además de otras explicadas en los párrafos a continuación.

Debajo del botón “Sincronizar” encontramos información sobre los módulos, como la calidad de conexión de radiofrecuencia LoRa, la fecha y la hora y los valores de los sensores de temperatura.

En la parte inferior encontramos el cuadro de mando, en donde podremos encender, apagar y programar las salidas. Dispone de las siguientes partes:

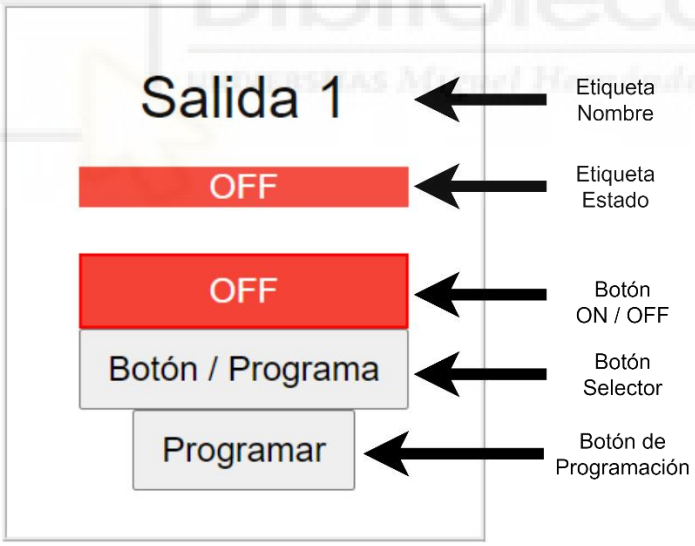


Figura 44: Detalle del panel de control de una salida

- **Etiqueta de nombre**

  Muestra de que salida se trata, incluyendo el número de la salida en la propia etiqueta.

- **Etiqueta de estado:**

  Tiene cuatro estados posibles:

- **OFF:** la salida se encuentra apagada y no se está utilizando el programa.
- **ON:** la salida se encuentra encendida y no se está utilizando el programa.
- **Programa OFF:** la salida se encuentra apagada y se está utilizando el programa.
- **Programa ON:** la salida se encuentra encendida y se está utilizando el programa.

- **Botón ON / OFF:**
  - Cuando no se está utilizando el programa, este botón servirá para encender o apagar la salida.
  - Cuando se esté utilizando el programa de la salida, esté botón quedará desactivado.
- **Botón Selector:**

  Este botón actúa como un *switch*, alternando entre el control de la salida por el botón ON / OFF o por el programa.
- **Botón de Programación:**

  Este programa nos redirigirá hacia la página de programación de la salida, desde la cual se podrá programar. La página tiene la siguiente estructura:

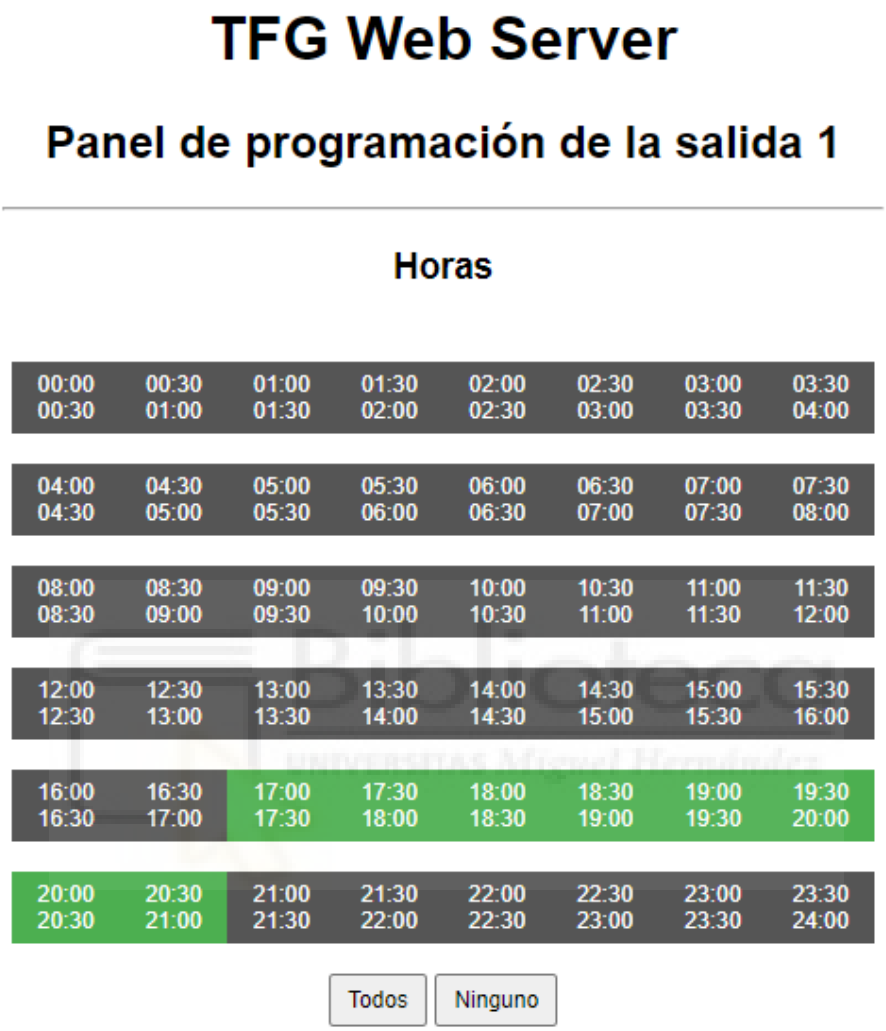


Figura 45: Página web de programación de las salidas (1 de 2)

Figura 46: Página web de programación de las salidas (2 de 2)

El ejemplo de programa mostrado en las figuras se activará desde las 17:00 a las 21:00, los fines de semana durante todo el año.

Si pulsamos el botón “Descartar” las modificaciones que hayamos realizado en la página se perderán.

Si pulsamos el botón “Programar” los cambios se almacenarán en memoria y se sincronizarán los datos entre el Módulo Servidor y el Módulo de Relés.

7. CONCLUSIONES

La realización de este TFG ha supuesto la puesta en práctica de muchos de los conceptos aprendidos en la grado de Ingeniería Electrónica y Automática Industrial, englobando campos como el diseño electrónico, diseño de placas de circuito impreso, automatización, programación, etc. Además ha servido para afianzar conceptos, sobre todo en el diseño electrónico y de PCBs, así como para profundizar en temas no tocados tan a fondo en el grado como la programación o las telecomunicaciones.

El haber realizado un producto desde su concepción hasta la obtención de un prototipo funcional hace entender la dificultad que esto supone: teniendo que decidir las capacidades del producto, ajustarlas a un presupuesto y a un tamaño concreto, fabricar un prototipo teniendo que soldar sus componentes y programar los microcontroladores, y por último, pero no por ello menos importante, testear el correcto funcionamiento de este.

Afrontar los fallos que se han ido dando durante el desarrollo del producto aporta una experiencia que no se puede adquirir únicamente con la teoría estudiada durante el grado, lo que le aporta aún más valor al trabajo llevado a cabo.

El prototipo desarrollado cumple con el objetivo de este TFG, que era desarrollar un prototipo de PLC para la automatización de bombas de piscina y de riego así como de electroválvulas. Junto con este, se ha desarrollado el código de programación necesario para su correcto funcionamiento.

El prototipo desarrollado cumple con los objetivos planteados:

- Controlable a través de internet.
- Largo alcance de la zona a automatizar y sin necesidad de cables, gracias a los módulos de radiofrecuencia LoRa.
- Potencia a manejar en las salidas de 5 A @ 250 VAC en las seis salidas genéricas y 17,5 A @ 250 VAC en las dos salidas dedicadas para motores.

- Posibilidad de añadir una sonda para medir la temperatura del agua de la piscina.

7.1. LÍNEAS FUTURAS

Con el objetivo de mejorar la funcionalidad del producto desarrollado para este TFG, se proponen las siguientes mejoras que harán al producto más funcional y atractivo para el mercado:

- Incluir los módulos en una caja de plástico, de forma que queden protegidos y mejore la estética del producto.
- Cifrado de los mensajes enviados por los módulos LoRa, de forma que no sea accesible su contenido a terceros.
- Optimizar el código de ambos módulos, de forma que aproveche al máximo las funcionalidad del microcontrolador ESP32, como por ejemplo su coprocesador de ultra bajo consumo.
- Añadir una pequeña pantalla LCD y unos botones de control al Módulo Servidor, de forma que se facilite el uso y la puesta en marcha de este por parte del consumidor. Gracias a esta pantalla se podría introducir las credenciales de la red wifi sin necesidad de modificar el código.
- Capacidad de ajuste de la fecha y hora del reloj interno del Módulo de Relés a través de la página web generada por el Módulo Servidor. De esta forma no haría falta modificar el código.
- Con las dos mejoras anteriores los módulos se podrían suministrar ya programados sin necesidad de que el cliente tenga que instalar ningún programa ni tenga que programar los módulos.
- Hacer que la dirección IP asignada por el rúter sea fija de forma que no haya que comprobar que IP se le ha asignado al módulo. Adicionalmente se podría visualizar la dirección IP asignada mediante una pantalla incluida en el Módulo Servidor.

8. BIBLIOGRAFÍA

[1] "Controlador lógico programable", *Wikipedia*, 2020. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Controlador_l%C3%B3gico_programable. [Accedido: 2-sep-2020]

[2] "Autómata PLC", *Google Shopping*, 2020. [En línea]. Disponible en: https://www.google.com/search?biw=1366&bih=625&tbm=shop&ei=vHloX5C1Bca8acTfmdAP&q=Aut%C3%B3mata+PLC. [Accedido: 4-sep-2020]

[3] "Reloj programador analógico", *Google Shopping*, 2020. [En línea]. Disponible en: https://www.google.com/search?biw=681&bih=611&tbm=shop&ei=QnhoX5TSGYeWa_L6msgN&q=Reloj+programador+anal%C3%B3gico&oq=Reloj+programador+anal%C3%B3gico. [Accedido: 4-sep-2020]

[4] "Reloj programador digital", *Google Shopping*, 2020. [En línea]. Disponible en: https://www.google.com/search?biw=1366&bih=625&tbm=shop&ei=t3hoX-KtOIuWa9XOhOgO&q=Reloj+programador+digital. [Accedido: 4-sep-2020]

[5] "Programadores y temporizadores", *ManoMano*, 2020. [En línea]. Disponible en: https://www.manomano.es/programadores-y-temporizadores-949. [Accedido: 4-sep-2020]

[6] P. Pickering, "Desarrollar con LoRa para aplicaciones IoT de baja tasa y largo alcance", *Digikey*, 2017. [En línea]. Disponible en: https://www.digikey.com/es/articles/develop-lora-for-low-rate-long-range-iot-applications. [Accedido: 4-sep-2020]

[7] "DS3231M ±5ppm, I2C Real-Time Clock", *Maxim Integrated Products Inc*, 2015. [En línea]. Disponible en: https://datasheets.maximintegrated.com/en/ds/DS3231M.pdf. [Accedido: 10-ago-2020]

[8] "TLE 8108 EM Smart 8-Channel Low Side Relay Driver with SPI Interface", *Infineon Technologies AG*, 2011. [En línea]. Disponible en:

https://www.infineon.com/dgdl/Infineon-TLE8108EM-DS-v01_00-en.pdf?fileId=db3a30433004641301302bf77ce863a2. [Accedido: 10-ago-2020]

[9] "DS18B20 Programmable Resolution 1-Wire Digital Thermometer", *Maxim Integrated Products Inc*, 2019. [En línea]. Disponible en: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf. [Accedido: 10-ago-2020]

[10] "ESP32-WROOM-32", *Espressif Systems*, 2019. [En línea]. Disponible en: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf. [Accedido: 10-ago-2020]

[11] "Ra-01 LoRa Module", *Ai-Thinker Technology Co.*, 2017. [En línea]. Disponible en: http://wiki.ai-thinker.com/_media/lora/docs/c047ps01a1_ra-01_product_specification_v1.1.pdf. [Accedido: 10-ago-2020]

[12] "Caja Estanca IP55 de Superficie con 7 Conos 113x113x60 mm", *Iluminashop*, 2020. [En línea]. Disponible en: https://iluminashop.com/led-producto/material-electrico/cajas-de-registro-y-empalme/caja-estanca-de-superficie-con-conos-113x113x60-mm. [Accedido: 12-sep-2020]

[13] "Relay ALQ (LQ) series", *Panasonic Electric Works*, 2012. [En línea]. Disponible en: https://media.digikey.com/pdf/Data%20Sheets/Panasonic%20Electric%20Works%20PDFs/ALQ(LQ)Relays_NewProdIntro.pdf. [Accedido: 21-sep-2020]

[14] "General Purpose Relays T9G series", *TE Connectivity Potter & Brumfield Relays*, 2020. [En línea]. Disponible en: https://www.te.com/commerce/DocumentDelivery/DDEController?Action=srchrtrv&DocNm=T9G_1216&DocType=DS&DocLang=English. [Accedido: 21-sep-2020]

[15] "ESP8266EX", *Espressif Systems*, 2020. [En línea]. Disponible en: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf. [Accedido: 20-sep-2020]

[16] "Qué es ESP8266", *Aprendiendo Arduino*, 2017. [En línea]. Disponible en: https://aprendiendoarduino.wordpress.com/2017/09/12/que-es-esp8266. [Accedido: 21-sep-2020]

[17] "ESP8266", *Wikipedia*, 2020. [En línea]. Disponible en: https://es.wikipedia.org/wiki/ESP8266. [Accedido: 21-sep-2020]

[18] "INSTALANDO EL ESP32", *Prometec*, 2018. [En línea]. Disponible en: https://www.prometec.net/instalando-esp32. [Accedido: 21-sep-2020]

[19] "ESP32", *Wikipedia*, 2020. [En línea]. Disponible en: https://es.wikipedia.org/wiki/ESP32. [Accedido: 21-sep-2020]

[20] "LoRa ra 01", *Aliexpress*, 2020. [En línea]. Disponible en: https://es.aliexpress.com/af/lora-ra%25252d01.html?d=y&origin=n&SearchText=lora+ra-01&catId=0&initiative_id=SB_20200921140656. [Accedido: 21-sep-2020]

[21] "Instant PCB Quote", *JLCPCB*, 2020. [En línea]. Disponible en: https://cart.jlcpcb.com/quote?orderType=1&stencilWidth=120&stencilLength=80. [Accedido: 22-sep-2020]

[22] R. Santos, "ESP32 Web Server – Arduino IDE", *Random Nerd Tutorials*, 2017. [En línea]. Disponible en: https://randomnerdtutorials.com/esp32-web-server-arduino-ide. [Accedido: 15-sep-2020]

[23] Sandeep Mistry, "LoRa", *GitHub*, 2020. [En línea]. Disponible en: https://github.com/sandeepmistry/arduino-LoRa. [Accedido: 13-sep-2020]

[24] "Xindar DP1000.031", *XINDAR*, 2020. [En línea]. Disponible en: https://xindar.com/producto/multimetro-digital-compacto-dp1000-031. [Accedido: 20-sep-2020]

[25] Adafruit, "RTClib", *GitHub*, 2020. [En línea]. Disponible en: https://github.com/adafruit/RTClib. [Accedido: 13-sep-2020]

[26] J. Studt, T. Pollard, R. James, P. Stoffregen, *et al.*, "OneWire", *PJRC*, 2020. [En línea]. Disponible en: https://www.pjrc.com/teensy/td_libs_OneWire.html. [Accedido: 13-sep-2020]

[27] M. Burton, T. Newsome, G. Barros y R. Tillaart, "DallasTemperature", *GitHub*, 2020. [En línea]. Disponible en: https://github.com/milesburton/Arduino-Temperature-Control-Library. [Accedido: 13-sep-2020]

[28] R. Santos, "Installing the ESP32 Board in Arduino IDE Windows", *Random Nerd Tutorials*, 2016. [En línea]. Disponible en: https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions. [Accedido: 15-sep-2020]

[29] R. Santos, "Installing the ESP32 Board in Arduino IDE Mac OS & Linux", *Random Nerd Tutorials*, 2016. [En línea]. Disponible en: https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-mac-and-linux-instructions. [Accedido: 15-sep-2020]

ANEXO I: CÓDIGO ARDUINO

1.1. CÓDIGO DEL MÓDULO SERVIDOR

```
/************
 * INCLUDES *
 ************/
#include <WiFi.h>   // To create the server and connect to the WiFi
#include <SPI.h>    // To communicate with the LoRa module
#include <LoRa.h>   // To send and receive messages with the LoRa module
#include <EEPROM.h> // To store the programming of the device


/**************
 * ESP32 PINS *
 **************/
//SPI
#define SPI_CLK 18
#define SPI_MISO 19
#define SPI_MOSI 23
//Define the pins used by the transceiver module
#define LORA_CS 5
#define LORA_RST 17
#define LORA_DIO0 16
//GPIOs for indications
#define LED_RED 4
#define LED_GREEN1 32
#define LED_GREEN2 33
#define LED_GREEN3 25
#define LED_GREEN4 26
#define LED_GREEN5 27
#define LED_GREEN6 12
#define LED_GREEN7 13
#define LED_GREEN8 2


/*************
 * CONSTANTS *
 *************/
#define EEPROM_SIZE 72    // Number of bytes that will be used to store the
programming
#define MESSAGE_LEN 85    // Number of bytes that the message will take
#define OFFSET_PROGRAM 13 // Where the programming starts in the message
#define SPI_CLK_FRQ 1E6   // SPI Clock (Máx 5MHz)
#define RESEND_ATTEMPTS 5 // Number of times a message will be sent while waiting for
confirmation
// Replace with your network credentials
const char* ssid = "ssid";
const char* password = "pass";
```

```
/********************
 * GLOBAL VARIABLES *
 ********************/
WiFiServer server(80);  // Set web server port number to 80
String header;          // Variable to store the HTTP request

// Variables to store the date
unsigned int time_seg, time_min, time_hour, time_day, time_month, time_year;
byte current_states = B00000000;  // Current states of the outputs
byte selector = B00000000;        // Selector between states and programming
byte programs[8][9] = {0};        // Variables to store the programming of the device
float temperature1, temperature2; // Variables to store the temperature of the
sensors
byte message[MESSAGE_LEN] = {0};  // Container for sending and receiving messages

byte programming = 0;             // Auxiliar for the html
int rssi = 0;                     // Auxiliar to store the RSSI of the message
received

// For server timing
unsigned long currentTime = millis();
unsigned long previousTime = 0;
const long timeoutTime = 2000;

// For send_data function timing
unsigned long startMillis;
unsigned long currentMillis;
const unsigned long period = 500;


/*************
 * FUNCTIONS *
 *************/
void print_time(){
  // Prints the time
  char time_buffer[26] = " ";
  char* formato = "%d/%02d/%02d - %02d:%02d:%02d";
  sprintf(time_buffer, formato, time_year, time_month, time_day, time_hour, time_min,
time_seg);
  Serial.print(time_buffer);
}

void print_message(){
  // Prints the message
  byte i;
  for(i = 0; i < MESSAGE_LEN; i++){
    Serial.print(i);
    Serial.print(":\t");
    Serial.println(message[i], BIN);
  }
}

float message_to_temperature(byte upper, byte lower){
```

```
  // Converts unsigned int 16 to float
  // 1  bit for the sign
  // 11 bits for the integer part
  // 4  bits for the decimal part
  unsigned int joined = 0;
  unsigned int decimal = 0;
  unsigned int integer = 0;
  float out;
  joined = upper;
  joined = joined << 8;
  joined = joined | lower;
  for(byte i = 0; i < 4; i++){
    bitWrite(decimal, i, bitRead(joined, i));
  }
  for(byte i = 0; i < 11; i++){
    bitWrite(integer, i, bitRead(joined, i + 4));
  }
  out = decimal;
  out /= 10;
  if (bitRead(joined, 15)){
    out = out * (-1) - integer;
  }else{
    out += integer;
  }
  return out;
}

byte weekday(int year, int month, int day){
  // Calculate day of week in proleptic Gregorian calendar. Sunday == 0
  int adjustment, mm, yy;
  if (year<2000) year+=2000;
  adjustment = (14 - month) / 12;
  mm = month + 12 * adjustment - 2;
  yy = year - adjustment;
  return (day + (13 * mm - 1) / 5 + yy + yy / 4 - yy / 100 + yy / 400) % 7;
}

byte calculate_outputs(){
  // Calculate wich states to switch on or off depending on the states and the
programming
  byte states = B00000000;
  byte i;
  byte aux_dayofweek;
  byte aux_minute = 0;
  for(i = 0; i < 8; i++){
    // Check the selector. If 1 then check time, if 0 then check state
    if (bitRead(selector, i)){
      // Check the month
      if (time_month < 9){
        if (!bitRead(EEPROM.read(i * 9 + 7), time_month - 1)){
          continue;
        }
      }else{
```

```
        if (!bitRead(EEPROM.read(i * 9 + 8), time_month - 9)){
          continue;
        }
      }
      // Check the week
      aux_dayofweek = weekday(time_year, time_month, time_day);
      if (!bitRead(EEPROM.read(i * 9 + 6), aux_dayofweek)){
        continue;
      }
      // Check the hour
      if (time_min >= 30){
          aux_minute = 1;
      }
      if (time_hour < 4){
        if (!bitRead(EEPROM.read(i * 9 + 0), (time_hour % 4) * 2 + aux_minute)){
          continue;
        }
      }else if (time_hour < 8){
        if (!bitRead(EEPROM.read(i * 9 + 1), (time_hour % 4) * 2 + aux_minute)){
          continue;
        }
      }else if (time_hour < 12){
        if (!bitRead(EEPROM.read(i * 9 + 2), (time_hour % 4) * 2 + aux_minute)){
          continue;
        }
      }else if (time_hour < 16){
        if (!bitRead(EEPROM.read(i * 9 + 3), (time_hour % 4) * 2 + aux_minute)){
          continue;
        }
      }else if (time_hour < 20){
        if (!bitRead(EEPROM.read(i * 9 + 4), (time_hour % 4) * 2 + aux_minute)){
          continue;
        }
      }else if (time_hour < 24){
        if (!bitRead(EEPROM.read(i * 9 + 5), (time_hour % 4) * 2 + aux_minute)){
          continue;
        }
      }
      bitWrite(states, i, 1);
    }else{
      bitWrite(states, i, bitRead(current_states, i));
    }
  }
  return states;
}

void message_to_variables(){
  // Update variables from the message
  byte i, j;
  time_seg = message[0];
  time_min = message[1];
  time_hour = message[2];
  time_day = message[3];
```

```
  time_month = message[4];
  time_year = message[5];
  time_year = time_year << 8;
  time_year = time_year | message[6];

  if (message[7] != current_states){
    Serial.println("We Should Update the relay states");
  }
  if (message[8] != selector){
    Serial.println("We Should Update the relay selector");
  }
  temperature1 = message_to_temperature(message[9], message[10]);
  temperature2 = message_to_temperature(message[11], message[12]);

  for(i = 0; i < 8; i++){
    for(j = 0; j < 9; j++){
      if (message[OFFSET_PROGRAM + i * 9 + j] != EEPROM.read(i * 9 + j)){
        Serial.println("We Should Update the relay programming");
      }
    }
  }
}

byte* variables_to_message(byte states){
  // Creates the message from the variables
  byte i, j;
  message[7] = states;
  message[8] = selector;
  for(i = 0; i < 8; i++){
    for(j = 0; j < 9; j++){
      message[OFFSET_PROGRAM + i * 9 + j] = EEPROM.read(i * 9 + j);
    }
  }
  return message;
}

void EEPROM_to_variables(){
  // Set variables with EEPROM values
  byte i, j;
  for (i = 0; i < 8; i++){
    for (j = 0; j < 9; j++){
      programs[i][j] = EEPROM.read(i * 9 + j);
    }
  }
}


void variables_to_EEPROM(){
  // Set EEPROM with variables values
  Serial.println("Escribiendo a memoria");
  byte i, j;
  for (i = 0; i < 8; i++){
    for (j = 0; j < 9; j++){
```

```
      EEPROM.write(i * 9 + j, programs[i][j]);
      EEPROM.commit();
    }
  }
}

int send_data(byte* message, unsigned int len){
  // Sends the message through LoRa and listens for the response
  byte flag = false;
  byte i, j;
  EEPROM_to_variables(); // Update the variables
  for (i = 0; i < RESEND_ATTEMPTS; i++){
    // Send LoRa packet to receiver
    digitalWrite(LED_RED, HIGH);
    LoRa.beginPacket();
    LoRa.write(message, len);
    LoRa.endPacket();
    Serial.println("Message sent");
    digitalWrite(LED_RED, LOW);

    // Wait for confirmation
    startMillis = millis();
    while(true){
      currentMillis = millis();
      // Check if too much time has elapsed
      if (currentMillis - startMillis >= period){
        break;
      }
      int packetSize = LoRa.parsePacket();
      if (packetSize) { // A packet has arrived
        j = 0;
        while (LoRa.available()) {  // There are still bytes to read from the packet
          if (j == MESSAGE_LEN - 1){
            flag = true;
          }else if(j >= MESSAGE_LEN){
            break;
          }
          message[j] = LoRa.read();
          j++;
        }
        // Print RSSI of packet
        rssi = LoRa.packetRssi();
        Serial.print("' with RSSI ");
        Serial.println(rssi);
        // If we have received all the message correctly
        if (flag){
          //print_message();
          // Update the variables
          message_to_variables();
          return 0;
        }
      }
      flag = false;
```

```
    }
  }
  return -1;
}

void update_outputs(){
  byte i;
  byte states = calculate_outputs();
  if (bitRead(states, 0)){
    digitalWrite(LED_GREEN1, HIGH);
  }else{
    digitalWrite(LED_GREEN1, LOW);
  }
  if (bitRead(states, 1)){
    digitalWrite(LED_GREEN2, HIGH);
  }else{
    digitalWrite(LED_GREEN2, LOW);
  }
  if (bitRead(states, 2)){
    digitalWrite(LED_GREEN3, HIGH);
  }else{
    digitalWrite(LED_GREEN3, LOW);
  }
  if (bitRead(states, 3)){
    digitalWrite(LED_GREEN4, HIGH);
  }else{
    digitalWrite(LED_GREEN4, LOW);
  }
  if (bitRead(states, 4)){
    digitalWrite(LED_GREEN5, HIGH);
  }else{
    digitalWrite(LED_GREEN5, LOW);
  }
  if (bitRead(states, 5)){
    digitalWrite(LED_GREEN6, HIGH);
  }else{
    digitalWrite(LED_GREEN6, LOW);
  }
  if (bitRead(states, 6)){
    digitalWrite(LED_GREEN7, HIGH);
  }else{
    digitalWrite(LED_GREEN7, LOW);
  }
  if (bitRead(states, 7)){
    digitalWrite(LED_GREEN8, HIGH);
  }else{
    digitalWrite(LED_GREEN8, LOW);
  }
}

void print_buttons(WiFiClient client, byte x){
  // Prints the buttons of the programming page
  byte h1, h2, m1, m2;
```

```
  char html_line[150]=" ";
  for (byte i = 0; i < 8; i++){
    h1 = 4 * x + i / 2;
    h2 = 4 * x + (i + 1) / 2;
    if (i % 2 == 0){
      m1 = 0;
      m2 = 30;
    }else{
      m1 = 30;
      m2 = 0;
    }
    if (bitRead(programs[programming][x], i)){
      char* formato = "<p><a href='/h/%i%i/off'><button style='width:12.5%%'
class='button_on'>%02d:%02d<br/>%02d:%02d</button></a></p>";
      sprintf(html_line, formato, x, i, h1, m1, h2, m2);
      client.println(html_line);
    }else{
      char* formato = "<p><a href='/h/%i%i/on'><button style='width:12.5%%'
class='button_off'>%02d:%02d<br/>%02d:%02d</button></a></p>";
      sprintf(html_line, formato, x, i, h1, m1, h2, m2);
      client.println(html_line);
    }
  }
}

void print_outputs(WiFiClient client, byte group){
  client.println("        <div class='btn-group' style='border-
style:ridge;padding:0px 25px'>");

  char html_line1[150]=" ";
  char* formato1 = "<p style='font-size: 20px; width:100%%; height:15px;'>Salida
%i</p>";
  sprintf(html_line1, formato1, group + 1);
  client.println(html_line1);

  if (bitRead(calculate_outputs(), group)){
    if (bitRead(selector, group)){
      client.println("          <p style='font-size: 12px; background-
color:#4CAF50;color: white;'>Programa: ON</p>");
    }else{
      client.println("          <p style='font-size: 12px; background-
color:#4CAF50;color: white;'>ON</p>");
    }
  }else{
    if (bitRead(selector, group)){
      client.println("          <p style='font-size: 12px; background-
color:#F44336;color: white;'>Programa: OFF</p>");
    }else{
      client.println("          <p style='font-size: 12px; background-
color:#F44336;color: white;'>OFF</p>");
    }
  }
  if (bitRead(selector, group)){
```

```
    if (bitRead(current_states, group)){
      client.println("          <p><a href='/'><button style='width:100%; border: 1px
solid green' class='button_blocked'>ON</button></a></p>");
    }else{
      client.println("          <p><a href='/'><button style='width:100%; border: 1px
solid red' class='button_blocked'>OFF</button></a></p>");
    }
  }else{
    if (bitRead(current_states, group)){
      char html_line2[150]=" ";
      char* formato2 = "<p><a href='/but/%i/off'><button style='width:100%%; border:
1px solid green' class='button_on'>ON</button></a></p>";
      sprintf(html_line2, formato2, group);
      client.println(html_line2);
    }else{
      char html_line3[150]=" ";
      char* formato3 = "<p><a href='/but/%i/on'><button style='width:100%%; border:
1px solid red' class='button_off'>OFF</button></a></p>";
      sprintf(html_line3, formato3, group);
      client.println(html_line3);
    }
  }
  if (bitRead(selector, group)){
    char html_line4[150]=" ";
    char* formato4 = "<p><a href='/sel/%i/off'><button style='float: left;
width:100%%;' class='btn btn-default col-md-4'>Bot&oacute;n /
Programa</button></a></p>";
    sprintf(html_line4, formato4, group);
    client.println(html_line4);
  }else{
    char html_line5[150]=" ";
    char* formato5 = "<p><a href='/sel/%i/on'><button style='float: left;
width:100%%;' class='btn btn-default col-md-4'>Bot&oacute;n /
Programa</button></a></p>";
    sprintf(html_line5, formato5, group);
    client.println(html_line5);
  }
  char html_line6[150]=" ";
  char* formato6 = "<p><a href='/prg/%i'><button class='btn btn-default col-md-
4'>Programar</button></a></p>";
  sprintf(html_line6, formato6, group);
  client.println(html_line6);
  client.println("        </div>");
}


/*********
 * SETUP *
 *********/
void setup() {
  Serial.begin(115200);

  // Initialize EEPROM with predefined size
```

```
EEPROM.begin(EEPROM_SIZE);
delay(500);
EEPROM_to_variables();

// Initialize the outputs
pinMode(LED_RED, OUTPUT);
pinMode(LED_GREEN1, OUTPUT);
pinMode(LED_GREEN2, OUTPUT);
pinMode(LED_GREEN3, OUTPUT);
pinMode(LED_GREEN4, OUTPUT);
pinMode(LED_GREEN5, OUTPUT);
pinMode(LED_GREEN6, OUTPUT);
pinMode(LED_GREEN7, OUTPUT);
pinMode(LED_GREEN8, OUTPUT);
// Set outputs to LOW
digitalWrite(LED_RED, LOW);
digitalWrite(LED_GREEN1, LOW);
digitalWrite(LED_GREEN2, LOW);
digitalWrite(LED_GREEN3, LOW);
digitalWrite(LED_GREEN4, LOW);
digitalWrite(LED_GREEN5, LOW);
digitalWrite(LED_GREEN6, LOW);
digitalWrite(LED_GREEN7, LOW);
digitalWrite(LED_GREEN8, LOW);

// Initialize SPI
SPI.begin(SPI_CLK, SPI_MISO, SPI_MOSI);
Serial.println("SPI started");

// Initialize LoRa transceiver module
LoRa.setPins(LORA_CS, LORA_RST, LORA_DIO0);
LoRa.setSPIFrequency(SPI_CLK_FRQ);
while (!LoRa.begin(433E6)) {
  Serial.println("Starting LoRa failed!");
  delay(500);
}
Serial.println("LoRa begined");
// Change sync word to match the receiver
// The sync word assures you don't get LoRa messages from other LoRa transceivers
// ranges from 0-0xFF
LoRa.setSyncWord(0xAE);

// Connect to Wi-Fi network with SSID and password
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
// Print local IP address and start web server
Serial.println("");
Serial.print("Connected to ");
```

```
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();
  Serial.println("HTTP server started");
}


/********
 * LOOP *
 ********/
void loop(){
  byte aux_button;
  byte* aux_message;
  WiFiClient client = server.available();   // Listen for incoming clients
  if (client) {                             // If a new client connects,
    currentTime = millis();
    previousTime = currentTime;
    Serial.println("New Client.");          // print a message out in the serial port
    String currentLine = "";                // make a String to hold incoming data
from the client
    while (client.connected() && currentTime - previousTime <= timeoutTime) {  //
loop while the client's connected
      currentTime = millis();
      if (client.available()) {             // if there's bytes to read from the
client,
        char c = client.read();             // read a byte, then
        Serial.write(c);                    // print it out the serial monitor
        header += c;
        if (c == '\n') {                    // if the byte is a newline character
          // if the current line is blank, you got two newline characters in a row.
          // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
            // and a content-type so the client knows what's coming, then a blank
line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

            /*************************
             * Interpret the requests *
             *************************/
            int index = header.indexOf("\n");
            if (header.substring(5, index).indexOf("but/") >= 0){
              // If a button has been pressed
              if (bitRead(selector, header[9] - 48) == 0){
                // If the selector is in button mode
                if (header.substring(5, index).indexOf("on") >= 0){
                  // Send information
                  aux_button = 0x00;
```

```
                aux_message = variables_to_message(current_states |
bitWrite(aux_button, header[9] - 48, 1));
                send_data(aux_message, MESSAGE_LEN);
                // Change the state
                bitWrite(current_states, header[9] - 48, 1);
              }else if (header.substring(5, index).indexOf("off") >= 0){
                // Send information
                aux_button = 0xFF;
                aux_message = variables_to_message(current_states &
bitWrite(aux_button, header[9] - 48, 0));
                send_data(aux_message, MESSAGE_LEN);
                // Change the state
                bitWrite(current_states, header[9] - 48, 0);
              }
            }
          }else if (header.substring(5, index).indexOf("prg/") >= 0){
            programming = header[9] - 48;
          }else if (header.substring(0, index).indexOf("GET /h/") >= 0){
            if (header.substring(5, index).indexOf("all") >= 0){
              if (header.substring(5, index).indexOf("on") >= 0){
                for(byte i = 0; i < 6; i++){
                  programs[programming][i] = 0xFF;
                }
              }else if (header.substring(5, index).indexOf("off") >= 0){
                for(byte i = 0; i < 6; i++){
                  programs[programming][i] = 0x00;
                }
              }
            }else{
              if (header.substring(5, index).indexOf("on") >= 0){
                bitWrite(programs[programming][header[7] - 48], header[8] - 48, 1);
              }else if (header.substring(5, index).indexOf("off") >= 0){
                bitWrite(programs[programming][header[7] - 48], header[8] - 48, 0);
              }
            }
          }else if (header.substring(0, index).indexOf("GET /d/") >= 0){
            if (header.substring(5, index).indexOf("all") >= 0){
              if (header.substring(5, index).indexOf("on") >= 0){
                programs[programming][6] = 0xFF;
              }else if (header.substring(5, index).indexOf("off") >= 0){
                programs[programming][6] = 0x00;
              }
            }else{
              if (header.substring(5, index).indexOf("on") >= 0){
                bitWrite(programs[programming][header[7] - 48], header[8] - 48, 1);
              }else if (header.substring(5, index).indexOf("off") >= 0){
                bitWrite(programs[programming][header[7] - 48], header[8] - 48, 0);
              }
            }
          }else if (header.substring(0, index).indexOf("GET /m/") >= 0){
            if (header.substring(5, index).indexOf("all") >= 0){
              if (header.substring(5, index).indexOf("on") >= 0){
                programs[programming][7] = 0xFF;
```

```
                programs[programming][8] = 0xFF;
              }else if (header.substring(5, index).indexOf("off") >= 0){
                programs[programming][7] = 0x00;
                programs[programming][8] = 0x00;
              }
            }else{
              if (header.substring(5, index).indexOf("on") >= 0){
                bitWrite(programs[programming][header[7] - 48], header[8] - 48, 1);
              }else if (header.substring(5, index).indexOf("off") >= 0){
                bitWrite(programs[programming][header[7] - 48], header[8] - 48, 0);
              }
            }
          }else if (header.substring(0, index).indexOf("GET /sincronize") >= 0){
            send_data(variables_to_message(current_states), MESSAGE_LEN);
          }else if (header.substring(0, index).indexOf("GET /program") >= 0){
            variables_to_EEPROM();
            send_data(variables_to_message(current_states), MESSAGE_LEN);
          }else if (header.substring(0, index).indexOf("GET /discard") >= 0){
            EEPROM_to_variables();
          }else if (header.substring(0, index).indexOf("GET /sel/") >= 0){
            if (header.substring(5, index).indexOf("on") >= 0){
              bitWrite(selector, header[9] - 48, 1);
            }else if (header.substring(5, index).indexOf("off") >= 0){
              bitWrite(selector, header[9] - 48, 0);
            }
            send_data(variables_to_message(current_states), MESSAGE_LEN);
          }

          /**********************************
           * SHOW THE CORRESPONDING WEB PAGE *
           **********************************/
          // Show the programming page
          if (header.substring(0, index).indexOf("GET /prg/") >= 0 |
              header.substring(0, index).indexOf("GET /h/") >= 0 |
              header.substring(0, index).indexOf("GET /d/") >= 0 |
              header.substring(0, index).indexOf("GET /m/") >= 0) {
            client.println("<!Doctype html>");
            client.println("<html>");
            client.println("  <head>");
            client.println("    <meta name='viewport' content='width=device-width,
initial-scale=1'>");
            client.println("    <link rel='icon' href='data:,'>");
            client.println("    <style>");
            client.println("      html {");
            client.println("        font-family: Helvetica;");
            client.println("        display: inline-block;");
            client.println("        margin: 0px auto;");
            client.println("        text-align: center;");
            client.println("      }");
            client.println("      .btn-group button:not(:last-child) {");
            client.println("        border-right: none; /* Prevent double borders
*/");
            client.println("      }");
```

```
client.println("      /* Clear floats (clearfix hack) */");
client.println("      .btn-group:after {");
client.println("        content: "";");
client.println("        clear: both;");
client.println("        display: table;");
client.println("      }");
client.println("      /* Add a background color on hover */");
client.println("      .btn-group button:hover {");
client.println("        background-color: #3e8e41;");
client.println("      }");
client.println("      .btn-group button {");
client.println("        padding: 5px 8px;");
client.println("        font-size: 12px;");
client.println("      }");
client.println("      .button_on {");
client.println("        background-color: #4CAF50;");
client.println("        border: none;");
client.println("        color: white;");
client.println("        padding: 2px 5px;");
client.println("        text-decoration: none;");
client.println("        font-size: 12px;");
client.println("        margin: 0px;");
client.println("        cursor: pointer;");
client.println("        float: left;");
client.println("      }");
client.println("      .button_off {");
client.println("        background-color: #555555;");
client.println("        border: none;");
client.println("        color: white;");
client.println("        padding: 2px 5px;");
client.println("        text-decoration: none;");
client.println("        font-size: 12px;");
client.println("        margin: 0px;");
client.println("        cursor: pointer;");
client.println("        float: left;");
client.println("      }");
client.println("      .fila button {");
client.println("        padding: 5px 8px;");
client.println("        font-size: 12px;");
client.println("      }");
client.println("      .fila > * {");
client.println("        padding:0px;");
client.println("        display: inline-block;");
client.println("        text-align: center;");
client.println("      }");
client.println("      .matriz {");
client.println("        padding:5px;");
client.println("        text-align: center;");
client.println("      }");
client.println("    </style>");
client.println("  </head>");
client.println("  <body>");
```

```
            client.println("    <h1 style='text-align: center;'>TFG Web
Server</h1>");

            char html_line[150]=" ";
            char* formato = "<h2 style='text-align: center;'>Panel de
programaci&oacute;n de la salida %i</h2>";
            sprintf(html_line, formato, programming + 1);
            client.println(html_line);

            client.println("    <hr>");
            client.println("    <h3 style='text-align: center;'>Horas</h3>");
            client.println("    <div class='matriz'>");
            client.println("      <div class='btn-group' style='width:100%'>");
            print_buttons(client, 0);
            client.println("      </div>");
            client.println("      <div class='btn-group' style='width:100%'>");
            print_buttons(client, 1);
            client.println("      </div>");
            client.println("      <div class='btn-group' style='width:100%'>");
            print_buttons(client, 2);
            client.println("      </div>");
            client.println("      <div class='btn-group' style='width:100%'>");
            print_buttons(client, 3);
            client.println("      </div>");
            client.println("      <div class='btn-group' style='width:100%'>");
            print_buttons(client, 4);
            client.println("      </div>");
            client.println("      <div class='btn-group' style='width:100%'>");
            print_buttons(client, 5);
            client.println("      </div>");
            client.println("      <div class='fila'>");
            client.println("        <p><a href='/h/all/on'><button type='button'
class='btn btn-default col-md-4'>Todos</button></a></p>");
            client.println("        <p><a href='/h/all/off'><button type='button'
class='btn btn-default col-md-4'>Ninguno</button></a></p>");
            client.println("      </div>");
            client.println("    </div>");
            client.println("    <hr>");
            client.println("    <h3 style='text-align: center;'>D&iacute;as</h3>");
            client.println("    <div class='matriz'>");
            client.println("      <div class='btn-group' style='width:100%'>");
            if (bitRead(programs[programming][6], 1)){
              client.println("        <p><a href='/d/61/off'><button
style='width:25%' class='button_on'>Lunes</button></a></p>");
            }else{
              client.println("        <p><a href='/d/61/on'><button
style='width:25%' class='button_off'>Lunes</button></a></p>");
            }
            if (bitRead(programs[programming][6], 2)){
              client.println("        <p><a href='/d/62/off'><button
style='width:25%' class='button_on'>Martes</button></a></p>");
            }else{
```

```
              client.println("       <p><a href='/d/62/on'><button
style='width:25%' class='button_off'>Martes</button></a></p>");
            }
            if (bitRead(programs[programming][6], 3)){
              client.println("       <p><a href='/d/63/off'><button
style='width:25%' class='button_on'>Mi&eacute;rcoles</button></a></p>");
            }else{
              client.println("       <p><a href='/d/63/on'><button
style='width:25%' class='button_off'>Mi&eacute;rcoles</button></a></p>");
            }
            if (bitRead(programs[programming][6], 4)){
              client.println("       <p><a href='/d/64/off'><button
style='width:25%' class='button_on'>Jueves</button></a></p>");
            }else{
              client.println("       <p><a href='/d/64/on'><button
style='width:25%' class='button_off'>Jueves</button></a></p>");
            }
            client.println("     </div>");
            client.println("     <div class='btn-group' style='width:100%>");
            if (bitRead(programs[programming][6], 5)){
              client.println("       <p><a href='/d/65/off'><button
style='width:33.3%' class='button_on'>Viernes</button></a></p>");
            }else{
              client.println("       <p><a href='/d/65/on'><button
style='width:33.3%' class='button_off'>Viernes</button></a></p>");
            }
            if (bitRead(programs[programming][6], 6)){
              client.println("       <p><a href='/d/66/off'><button
style='width:33.3%' class='button_on'>S&aacute;bado</button></a></p>");
            }else{
              client.println("       <p><a href='/d/66/on'><button
style='width:33.3%' class='button_off'>S&aacute;bado</button></a></p>");
            }
            if (bitRead(programs[programming][6], 0)){
              client.println("       <p><a href='/d/60/off'><button
style='width:33.3%' class='button_on'>Domingo</button></a></p>");
            }else{
              client.println("       <p><a href='/d/60/on'><button
style='width:33.3%' class='button_off'>Domingo</button></a></p>");
            }
            client.println("     </div>");
            client.println("     <div class='fila'>");
            client.println("       <p><a href='/d/all/on'><button type='button'
class='btn btn-default col-md-4'>Todos</button></a></p>");
            client.println("       <p><a href='/d/all/off'><button type='button'
class='btn btn-default col-md-4'>Ninguno</button></a></p>");
            client.println("     </div>");
            client.println("   </div>");
            client.println("   <hr>");
            client.println("   <h3 style='text-align: center;'>Meses</h3>");
            client.println("   <div class='matriz'>");
            client.println("     <div class='btn-group' style='width:100%'>");
            if (bitRead(programs[programming][7], 0)){
```

```
                client.println("        <p><a href='/d/70/off'><button
style='width:25%' class='button_on'>Enero</button></a></p>");
              }else{
                client.println("        <p><a href='/d/70/on'><button
style='width:25%' class='button_off'>Enero</button></a></p>");
              }
              if (bitRead(programs[programming][7], 1)){
                client.println("        <p><a href='/d/71/off'><button
style='width:25%' class='button_on'>Febrero</button></a></p>");
              }else{
                client.println("        <p><a href='/d/71/on'><button
style='width:25%' class='button_off'>Febrero</button></a></p>");
              }
              if (bitRead(programs[programming][7], 2)){
                client.println("        <p><a href='/d/72/off'><button
style='width:25%' class='button_on'>Marzo</button></a></p>");
              }else{
                client.println("        <p><a href='/d/72/on'><button
style='width:25%' class='button_off'>Marzo</button></a></p>");
              }
              if (bitRead(programs[programming][7], 3)){
                client.println("        <p><a href='/d/73/off'><button
style='width:25%' class='button_on'>Abril</button></a></p>");
              }else{
                client.println("        <p><a href='/d/73/on'><button
style='width:25%' class='button_off'>Abril</button></a></p>");
              }
              client.println("      </div>");
              client.println("      <div class='btn-group' style='width:100%'>");
              if (bitRead(programs[programming][7], 4)){
                client.println("        <p><a href='/d/74/off'><button
style='width:25%' class='button_on'>Mayo</button></a></p>");
              }else{
                client.println("        <p><a href='/d/74/on'><button
style='width:25%' class='button_off'>Mayo</button></a></p>");
              }
              if (bitRead(programs[programming][7], 5)){
                client.println("        <p><a href='/d/75/off'><button
style='width:25%' class='button_on'>Junio</button></a></p>");
              }else{
                client.println("        <p><a href='/d/75/on'><button
style='width:25%' class='button_off'>Junio</button></a></p>");
              }
              if (bitRead(programs[programming][7], 6)){
                client.println("        <p><a href='/d/76/off'><button
style='width:25%' class='button_on'>Julio</button></a></p>");
              }else{
                client.println("        <p><a href='/d/76/on'><button
style='width:25%' class='button_off'>Julio</button></a></p>");
              }
              if (bitRead(programs[programming][7], 7)){
                client.println("        <p><a href='/d/77/off'><button
style='width:25%' class='button_on'>Agosto</button></a></p>");
```

```
            }else{
              client.println("       <p><a href='/d/77/on'><button
style='width:25%' class='button_off'>Agosto</button></a></p>");
            }
            client.println("     </div>");
            client.println("     <div class='btn-group' style='width:100%'>");
            if (bitRead(programs[programming][8], 0)){
              client.println("       <p><a href='/d/80/off'><button
style='width:25%' class='button_on'>Septiembre</button></a></p>");
            }else{
              client.println("       <p><a href='/d/80/on'><button
style='width:25%' class='button_off'>Septiembre</button></a></p>");
            }
            if (bitRead(programs[programming][8], 1)){
              client.println("       <p><a href='/d/81/off'><button
style='width:25%' class='button_on'>Octubre</button></a></p>");
            }else{
              client.println("       <p><a href='/d/81/on'><button
style='width:25%' class='button_off'>Octubre</button></a></p>");
            }
            if (bitRead(programs[programming][8], 2)){
              client.println("       <p><a href='/d/82/off'><button
style='width:25%' class='button_on'>Noviembre</button></a></p>");
            }else{
              client.println("       <p><a href='/d/82/on'><button
style='width:25%' class='button_off'>Noviembre</button></a></p>");
            }
            if (bitRead(programs[programming][8], 3)){
              client.println("       <p><a href='/d/83/off'><button
style='width:25%' class='button_on'>Diciembre</button></a></p>");
            }else{
              client.println("       <p><a href='/d/83/on'><button
style='width:25%' class='button_off'>Diciembre</button></a></p>");
            }
            client.println("     </div>");
            client.println("     <div class='fila'>");
            client.println("       <p><a href='/m/all/on'><button type='button'
class='btn btn-default col-md-4'>Todos</button></a></p>");
            client.println("       <p><a href='/m/all/off'><button type='button'
class='btn btn-default col-md-4'>Ninguno</button></a></p>");
            client.println("     </div>");
            client.println("   </div>");
            client.println("   <hr>");
            client.println("   <div class='matriz'>");
            client.println("     <div class='fila'>");
            client.println("       <p><a href='/program'><button type='button'
class='btn btn-default col-md-4'>Programar</button></a></p>");
            client.println("       <p><a href='/discard'><button type='button'
class='btn btn-default col-md-4'>Descartar</button></a></p>");
            client.println("     </div>");
            client.println("   </div>");
            client.println("  </body>");
            client.println("</html>");
```

```
            }

            // Show the main page
            else{
              client.println("<!Doctype html>");
              client.println("<html>");
              client.println("  <head>");
              client.println("    <meta name='viewport' content='width=device-width,
initial-scale=1'>");
              client.println("    <link rel='icon' href='data:,'>");
              client.println("    <style>");
              client.println("      html {");
              client.println("        font-family: Helvetica;");
              client.println("        display: inline-block;");
              client.println("        margin: 0px auto;");
              client.println("        text-align: center;");
              client.println("      }");
              client.println("      .btn-group button:not(:last-child) {");
              client.println("        border-right: none; /* Prevent double borders
*/");
              client.println("      }");
              client.println("      /* Clear floats (clearfix hack) */");
              client.println("      .btn-group:after {");
              client.println("        content: "";");
              client.println("        clear: both;");
              client.println("        display: table;");
              client.println("      }");
              client.println("      .btn-group button {");
              client.println("        padding: 5px 8px;");
              client.println("        font-size: 12px;");
              client.println("      }");
              client.println("      .button_on {");
              client.println("        background-color: #4CAF50;");
              client.println("        border: none;");
              client.println("        color: white;");
              client.println("        padding: 2px 5px;");
              client.println("        text-decoration: none;");
              client.println("        font-size: 12px;");
              client.println("        margin: 0px;");
              client.println("        cursor: pointer;");
              client.println("        float: left;");
              client.println("      }");
              client.println("      .button_off {");
              client.println("        background-color: #F44336;");
              client.println("        border: none;");
              client.println("        color: white;");
              client.println("        padding: 2px 5px;");
              client.println("        text-decoration: none;");
              client.println("        font-size: 12px;");
              client.println("        margin: 0px;");
              client.println("        cursor: pointer;");
              client.println("        float: left;");
              client.println("      }");
```

```
            client.println("      .button_blocked {");
            client.println("        background-color: #616161;");
            client.println("        border: none;");
            client.println("        color: white;");
            client.println("        padding: 2px 5px;");
            client.println("        text-decoration: none;");
            client.println("        font-size: 12px;");
            client.println("        margin: 0px;");
            client.println("        cursor: pointer;");
            client.println("        float: left;");
            client.println("      }");
            client.println("      .fila button {");
            client.println("        padding: 5px 8px;");
            client.println("        font-size: 12px;");
            client.println("      }");
            client.println("      .fila > * {");
            client.println("        display: inline-block;");
            client.println("        text-align: center;");
            client.println("      }");
            client.println("      .matriz {");
            client.println("        padding:5px;");
            client.println("        text-align: center;");
            client.println("      }");
            client.println("      .btn btn-default col-md-4 {");
            client.println("        border: none;");
            client.println("        color: white;");
            client.println("        padding: 2px 5px;");
            client.println("        text-decoration: none;");
            client.println("        font-size: 12px;");
            client.println("        margin: 0px;");
            client.println("        cursor: pointer;");
            client.println("        float: left;");
            client.println("      }");
            client.println("    </style>");
            client.println("  </head>");
            client.println("  <body>");
            client.println("    <h1 style='text-align: center;'>TFG Web
Server</h1>");
            client.println("    <h2 style='text-align: center;'>Panel de control de
las salidas</h2>");
            client.println("    <p><a href='/sincronize'><button class='btn btn-
default col-md-4'>Sincronizar</button></a></p>");

            char time_buffer[150] = " ";
            char* formato = "<p style='font-size: 14px; width:100%%;
height:5px;'>%04d/%02d/%02d - %02d:%02d:%02d</p>";
            sprintf(time_buffer, formato, time_year, time_month, time_day,
time_hour, time_min, time_seg);
            client.println(time_buffer);

            client.println("    <div class='fila'>");
            client.println("      <div class='btn-group' style='border-
style:ridge;'>");
```

```
            client.println("       <p style='font-size: 14px; width:100%;
height:5px;'>Temperatura Rel&eacute;s</p>");

            char html_line1[150]=" ";
            char* formato1 = "<p style='font-size: 14px; width:100%%;
height:5px;'>%.1f</p>";
            sprintf(html_line1, formato1, temperature1);
            client.println(html_line1);

            client.println("     </div>");
            client.println("     <div class='btn-group' style='border-
style:ridge;'>");
            client.println("       <p style='font-size: 14px; width:100%;
height:5px;'>Temperatura Sonda</p>");

            char html_line2[150]=" ";
            char* formato2 = "<p style='font-size: 14px; width:100%%;
height:5px;'>%.1f</p>";
            sprintf(html_line2, formato2, temperature2);
            client.println(html_line2);

            client.println("     </div>");
            client.println("   </div>");

            char html_line3[150]=" ";
            if (rssi == 0){
              client.println("<p style='font-size: 14px'>RSSI: 0</p>");
            }else if (rssi > -70){
              char* formato3 = "<p style='font-size: 14px; background-color:green;
color: white'>RSSI: %i</p>";
              sprintf(html_line3, formato3, rssi);
              client.println(html_line3);
            }else if (rssi > -85){
              char* formato3 = "<p style='font-size: 14px; background-
color:yellow'>RSSI: %i</p>";
              sprintf(html_line3, formato3, rssi);
              client.println(html_line3);
            }else if (rssi > -100){
              char* formato3 = "<p style='font-size: 14px; background-
color:red'>RSSI: %i</p>";
              sprintf(html_line3, formato3, rssi);
              client.println(html_line3);
            }else if (rssi > -150){
              char* formato3 = "<p style='font-size: 14px; background-color:black;
color: white;'>RSSI: %i</p>";
              sprintf(html_line3, formato3, rssi);
              client.println(html_line3);
            }

            client.println("    <hr>");
            client.println("    <h3 style='text-align: center;'>Rel&eacute;s de
baja potencia</h3>");
            client.println("    <div class='matriz'>");
```

```
            client.println("      <div class='fila'>");
            print_outputs(client, 0);
            print_outputs(client, 1);
            client.println("      </div>");
            client.println("      <div class='fila'>");
            print_outputs(client, 2);
            print_outputs(client, 3);
            client.println("      </div>");
            client.println("      <div class='fila'>");
            print_outputs(client, 4);
            print_outputs(client, 5);
            client.println("      </div>");
            client.println("    </div>");
            client.println("    <hr>");
            client.println("    <h3 style='text-align: center;'>Rel&eacute;s para
motores</h3>");
            client.println("    <div class='matriz'>");
            client.println("      <div class='fila'>");
            print_outputs(client, 6);
            print_outputs(client, 7);
            client.println("      </div>");
            client.println("    </div>");
            client.println("  </body>");
            client.println("</html>");
          }


          // The HTTP response ends with another blank line
          client.println();
          // Break out of the while loop
          break;
        } else { // if you got a newline, then clear currentLine
          currentLine = "";
        }
      } else if (c != '\r') {  // if you got anything else but a carriage return
character,
        currentLine += c;      // add it to the end of the currentLine
      }
    }
  }
  // Clear the header variable
  header = "";
  // Close the connection
  client.stop();
  Serial.println("Client disconnected.");
  Serial.println("");
 }
 update_outputs();
}
```

1.2. CÓDIGO DEL MÓDULO DE RELÉS

```
/************
 * INCLUDES *
 ************/
#include <SPI.h>                // To communicate with the relay driver and LoRa
module
#include <LoRa.h>               // To send and receive messages with the LoRa module
#include <Wire.h>               // To communicate with the DS3231 (I2C)
#include <RTClib.h>             // To manage the DS3231
#include <EEPROM.h>             // To store the programming of the device
// To read the temperature of the probe
#include <OneWire.h>
#include <DallasTemperature.h>


/**************
 * ESP32 PINS *
 **************/
// SPI
#define SPI_CLK 18
#define SPI_MISO 19
#define SPI_MOSI 23
// LoRa pins
#define LORA_CS 5
#define LORA_RST 17
#define LORA_DIO0 16
// Relay driver
#define DRIVER_RST 32
#define DRIVER_CS 33
// I2C
#define SCL 22
#define SDA 21
// RTC
#define RTC_INT 25
#define RTC_RST 27
// GPIOs for indications
#define LED_RED 4
// Temperature probe
#define TEMP_PROBE 26


/*************
 * CONSTANTS *
 *************/
#define EEPROM_SIZE 72    // Number of bytes that will be used to store the
programming
#define MESSAGE_LEN 85    // Number of bytes that the message will take
#define OFFSET_PROGRAM 13 // Where the programming starts in the message
#define SPI_CLK_FRQ 1E6   // SPI Clock (Máx 5MHz)
```

```
/********************
 * GLOBAL VARIABLES *
 ********************/
RTC_DS3231 rtc;
byte current_states = B00000000;  // Current states of the outputs
byte selector = B00000000;        // Selector between states and programming
byte message[MESSAGE_LEN] = {0};  // Container for sending and receiving messages
// To read the temperature of the probe
OneWire oneWireBus(TEMP_PROBE);
DallasTemperature sensor(&oneWireBus);


/*************
 * FUNCTIONS *
 *************/
void print_time(){
  // Prints the time from the RTC
  DateTime fecha = rtc.now();
  char time_buffer[26] = " ";
  char* formato = "%d/%02d/%02d - %02d:%02d:%02d";
  sprintf(time_buffer, formato, fecha.year(), fecha.month(), fecha.day(),
          fecha.hour(), fecha.minute(), fecha.second());
  Serial.print(time_buffer);
}

void print_message(){
  // Prints the message
  byte i;
  for(i = 0; i < MESSAGE_LEN; i++){
    Serial.print(i);
    Serial.print(":\t");
    Serial.println(message[i], BIN);
  }
}

unsigned int temperature_to_message(float value){
  // Converts the float temperature to unsigned int 16
  // 1  bit for the sign
  // 11 bits for the integer part
  // 4  bits for the decimal part
  unsigned int out = 0x0000;
  int aux_int = (int)value;
  unsigned int aux_uint;
  unsigned int aux_decimal = (value - aux_int) * 10;
  for(byte i = 0; i < 4; i++){
    bitWrite(out, i, bitRead(aux_decimal, i));
  }
  if (value < 0){
    bitWrite(out, 15, 1);
    aux_uint = aux_int * (-1);
  }else{
    aux_uint = aux_int;
  }
```

```
  for(byte i = 0; i < 11; i++){
    bitWrite(out, i + 4, bitRead(aux_uint, i));
  }
  return out;
}

unsigned int states_to_TLE8108(byte states){
  // Converts the given states to TLE8108 SPI protocol
  // 0 --> 11   OFF state
  // 1 --> 10    ON state
  unsigned int out = 0x0000;
  byte aux = B00000001;
  byte i;
  for(i = 0; i < 8; i++){
    if (((states & aux) >> i) == 1){
      out = out | (B10 << (i * 2));
    }else{
      out = out | (B11 << (i * 2));
    }
    aux = aux << 1;
  }
  return(out);
}

byte weekday(int year, int month, int day){
  // Calculate day of week in proleptic Gregorian calendar. Sunday == 0
  int adjustment, mm, yy;
  if (year<2000) year+=2000;
  adjustment = (14 - month) / 12;
  mm = month + 12 * adjustment - 2;
  yy = year - adjustment;
  return (day + (13 * mm - 1) / 5 + yy + yy / 4 - yy / 100 + yy / 400) % 7;
}

byte calculate_outputs(){
  // Calculate wich states to switch on or off depending on the states and the
programming
  byte states = B00000000;
  byte i;
  byte aux_dayofweek;
  byte aux_minute = 0;
  DateTime fecha = rtc.now();
  for(i = 0; i < 8; i++){
    // Check the selector. If 1 then check time, if 0 then check state
    if (bitRead(selector, i)){
      // Check the month
      if (fecha.month() < 9){
        if (!bitRead(EEPROM.read(i * 9 + 7), fecha.month() - 1)){
          continue;
        }
      }else{
        if (!bitRead(EEPROM.read(i * 9 + 8), fecha.month() - 9)){
          continue;
```

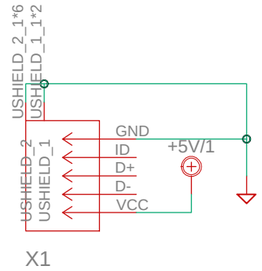
```
        }
      }
      // Check the week
      aux_dayofweek = weekday(fecha.year(), fecha.month(), fecha.day());
      if (!bitRead(EEPROM.read(i * 9 + 6), aux_dayofweek)){
        continue;
      }
      // Check the hour
      if (fecha.minute() >= 30){
          aux_minute = 1;
      }
      if (fecha.hour() < 4){
        if (!bitRead(EEPROM.read(i * 9 + 0), (fecha.hour() % 4) * 2 + aux_minute)){
          continue;
        }
      }else if (fecha.hour() < 8){
        if (!bitRead(EEPROM.read(i * 9 + 1), (fecha.hour() % 4) * 2 + aux_minute)){
          continue;
        }
      }else if (fecha.hour() < 12){
        if (!bitRead(EEPROM.read(i * 9 + 2), (fecha.hour() % 4) * 2 + aux_minute)){
          continue;
        }
      }else if (fecha.hour() < 16){
        if (!bitRead(EEPROM.read(i * 9 + 3), (fecha.hour() % 4) * 2 + aux_minute)){
          continue;
        }
      }else if (fecha.hour() < 20){
        if (!bitRead(EEPROM.read(i * 9 + 4), (fecha.hour() % 4) * 2 + aux_minute)){
          continue;
        }
      }else if (fecha.hour() < 24){
        if (!bitRead(EEPROM.read(i * 9 + 5), (fecha.hour() % 4) * 2 + aux_minute)){
          continue;
        }
      }
      bitWrite(states, i, 1);
    }else{
      bitWrite(states, i, bitRead(current_states, i));
    }
  }
  return states;
}

unsigned int relay_switcher(byte states){
  // Send the states to the relay driver
  unsigned int driver_out;
  SPI.beginTransaction(SPISettings(SPI_CLK_FRQ, MSBFIRST, SPI_MODE1));
  digitalWrite(DRIVER_CS, LOW);
  driver_out = SPI.transfer16(states_to_TLE8108(states));
  digitalWrite(DRIVER_CS, HIGH);
  SPI.endTransaction();
  // Return information given by the relay driver
```

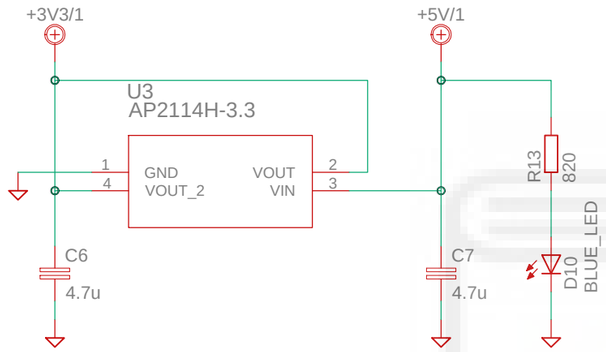
```
  return driver_out;
}

void message_to_variables(){
  // Update variables from the message
  byte i, j;
  current_states = message[7];
  selector = message[8];
  for(i = 0; i < 8; i++){
    for(j = 0; j < 9; j++){
      EEPROM.write(i * 9 + j, message[OFFSET_PROGRAM + i * 9 + j]);
      EEPROM.commit();
    }
  }
}

byte* variables_to_message(byte states){
  // Creates the message from the variables
  DateTime fecha = rtc.now();
  unsigned int temp;
  message[0] = lowByte(fecha.second());
  message[1] = lowByte(fecha.minute());
  message[2] = lowByte(fecha.hour());
  message[3] = lowByte(fecha.day());
  message[4] = lowByte(fecha.month());
  message[5] = lowByte(fecha.year() >> 8);
  message[6] = lowByte(fecha.year());
  message[7] = states;
  message[8] = selector;
  temp = temperature_to_message(rtc.getTemperature());
  message[9] = lowByte(temp >> 8);
  message[10] = lowByte(temp);
  sensor.requestTemperatures();
  temp = temperature_to_message(sensor.getTempCByIndex(0));
  message[11] = lowByte(temp >> 8);
  message[12] = lowByte(temp);
  for(byte i = 0; i < 8; i++){
    for(byte j = 0; j < 9; j++){
      message[OFFSET_PROGRAM + i * 9 + j] = EEPROM.read(i * 9 + j);
    }
  }
  return message;
}

void send_data(byte* message, unsigned int len){
  // Send the message
  digitalWrite(LED_RED, HIGH);
  LoRa.beginPacket();
  LoRa.write(message, len);
  LoRa.endPacket();
  Serial.println("Message sent");
  digitalWrite(LED_RED, LOW);
}
```

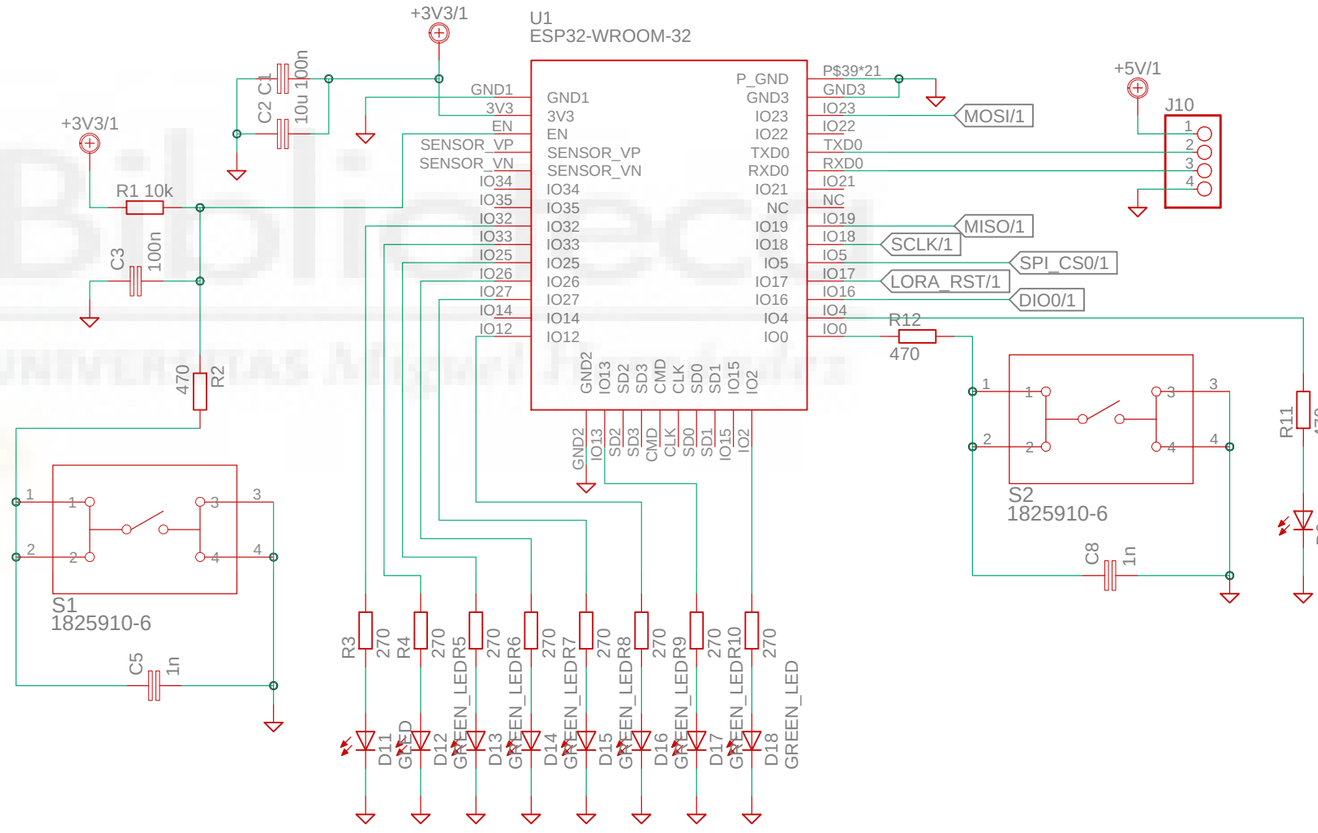
```
/*********
 * SETUP *
 *********/
void setup() {
  Serial.begin(115200);

  // Initialize EEPROM with predefined size
  EEPROM.begin(EEPROM_SIZE);
  delay(500);

  // Initialize the outputs
  pinMode(LED_RED, OUTPUT);
  digitalWrite(LED_RED, LOW);
  pinMode(DRIVER_RST, OUTPUT);
  digitalWrite(DRIVER_RST, HIGH);
  pinMode(DRIVER_CS, OUTPUT);
  digitalWrite(DRIVER_CS, HIGH);

  // Initialize SPI
  SPI.begin(SPI_CLK, SPI_MISO, SPI_MOSI);
  Serial.println("SPI started");

  // Initialize LoRa transceiver module
  LoRa.setPins(LORA_CS, LORA_RST, LORA_DIO0);
  LoRa.setSPIFrequency(SPI_CLK_FRQ);
  while (!LoRa.begin(433E6)) {
    Serial.println("Starting LoRa failed!");
    delay(500);
  }
  Serial.println("LoRa begined");
  // Change sync word to match the receiver
  // The sync word assures you don't get LoRa messages from other LoRa transceivers
  // ranges from 0-0xFF
  LoRa.setSyncWord(0xAE);

  // Initialize the RTC
  if (!rtc.begin()){
    Serial.println("RTC module not found");
    while(1);
  }
  // Update the RTC time
  rtc.adjust(DateTime(__DATE__, __TIME__));

  // Initialize the temperature probe
  sensor.begin();
}


/********
 * LOOP *
 ********/
```

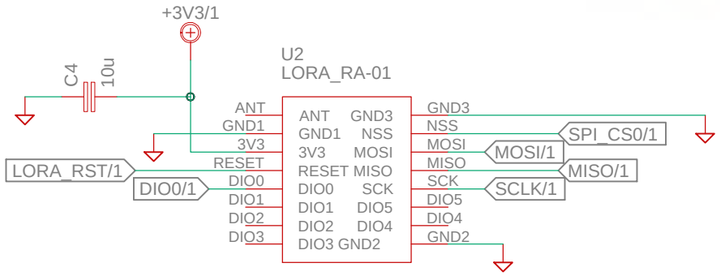
```
void loop() {
  byte flag = false;
  byte i = 0;
  unsigned int relay_driver_out;
  // Try to parse packet
  int packetSize = LoRa.parsePacket();
  if (packetSize) { // A packet has arrived
    digitalWrite(LED_RED, HIGH);
    Serial.print("Message received");
    while (LoRa.available()) {  // There are still bytes to read from the packet
      if (i == MESSAGE_LEN - 1){
        flag = true;
      }else if(i >= MESSAGE_LEN){
        break;
      }
      message[i] = LoRa.read();
      i++;
    }
    // Print RSSI of packet
    Serial.print(" with RSSI ");
    Serial.println(LoRa.packetRssi());
    digitalWrite(LED_RED, LOW);
    // If we have received all the message correctly
    if (flag){
      //print_message();
      // Update the variables
      message_to_variables();
      // Check if we need to turn on or off the outputs
      relay_driver_out = relay_switcher(calculate_outputs());
      if (relay_driver_out != 0xFFFF){
        Serial.print("WARNING - Relay Driver Response: ");
        Serial.println(relay_driver_out, BIN);
      }
      // Send confirmation to the server
      send_data(variables_to_message(current_states), MESSAGE_LEN);
    }
  }
  // Check if we need to turn on or off the outputs
  relay_driver_out = relay_switcher(calculate_outputs());
  if (relay_driver_out != 0xFFFF){
    Serial.print("WARNING - Relay Driver Response: ");
    Serial.println(relay_driver_out, BIN);
  }
}
```

ANEXO II: PLANOS

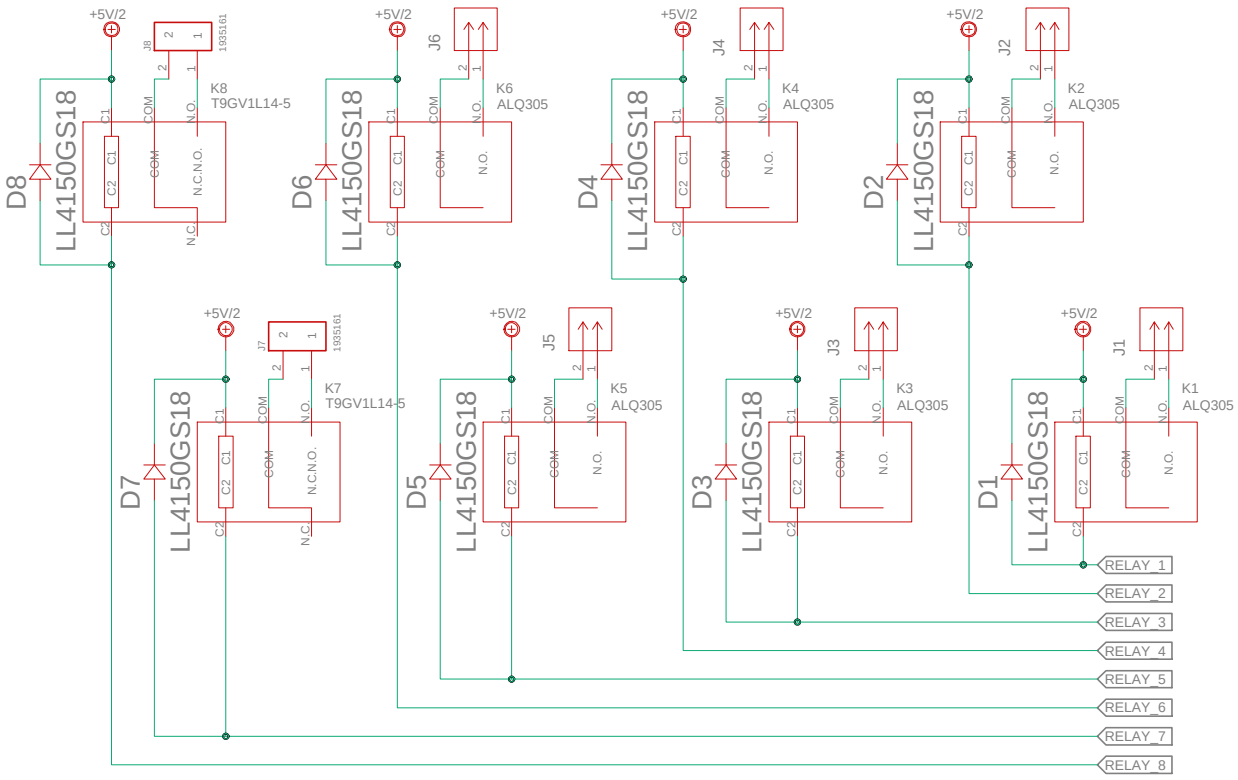


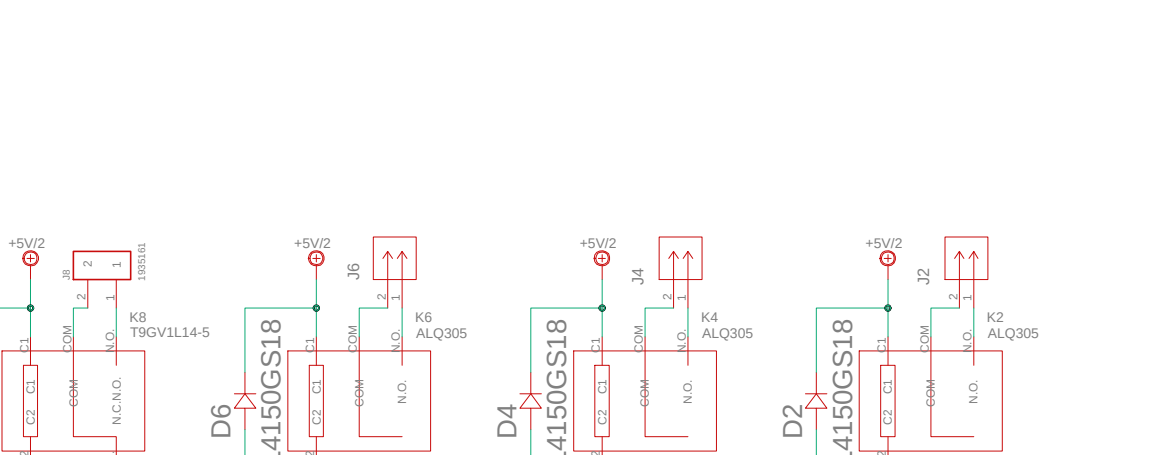


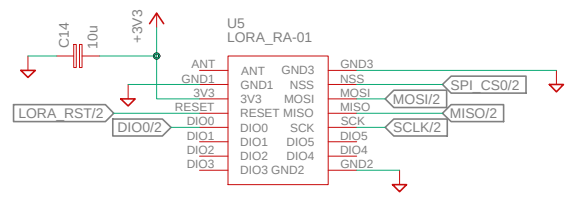


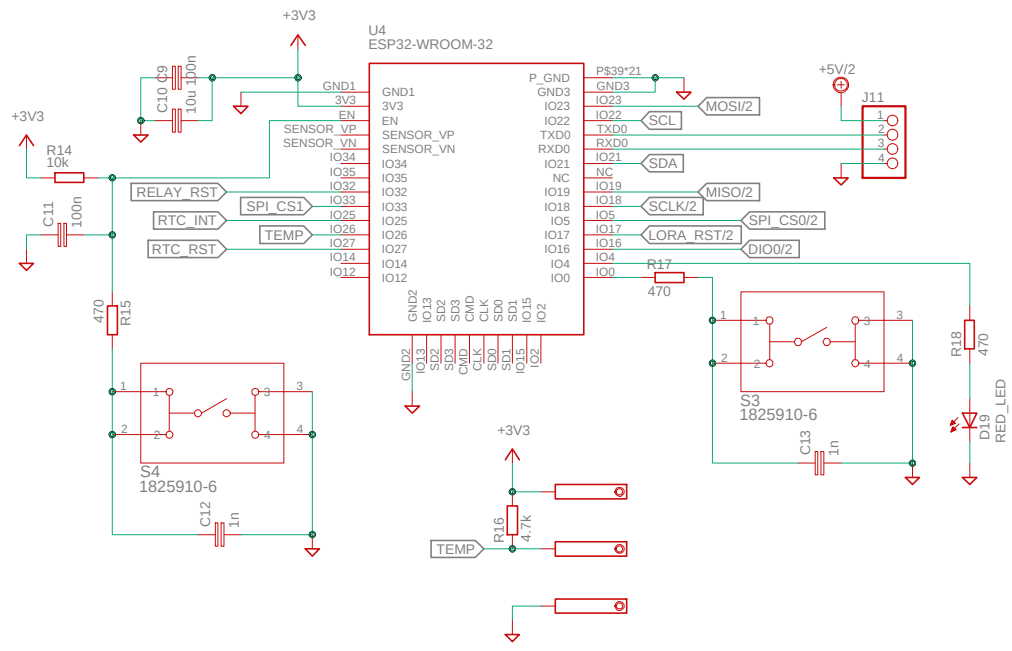


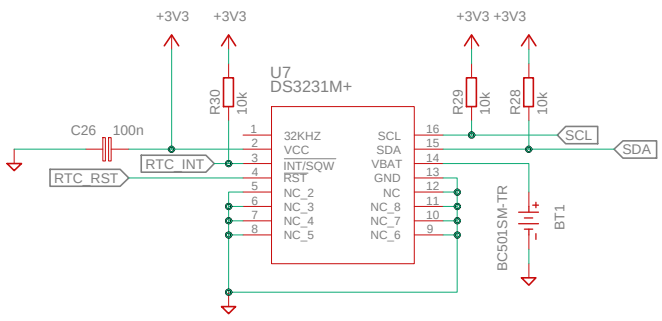
Escala	-	Proyectante	Juan Luis Leal Contreras	Fecha
TFG - PLC PARA CONTROL DE BOMBAS DE PISCINA Y SISTEMA DE RIEGO AUTOMÁTICO EN CHALÉS		Tutor	David Marroquí Sempere	22/09/2020
		ESQUEMÁTICO MÓDULO SERVIDOR		Nº Hoja: 1 de 4



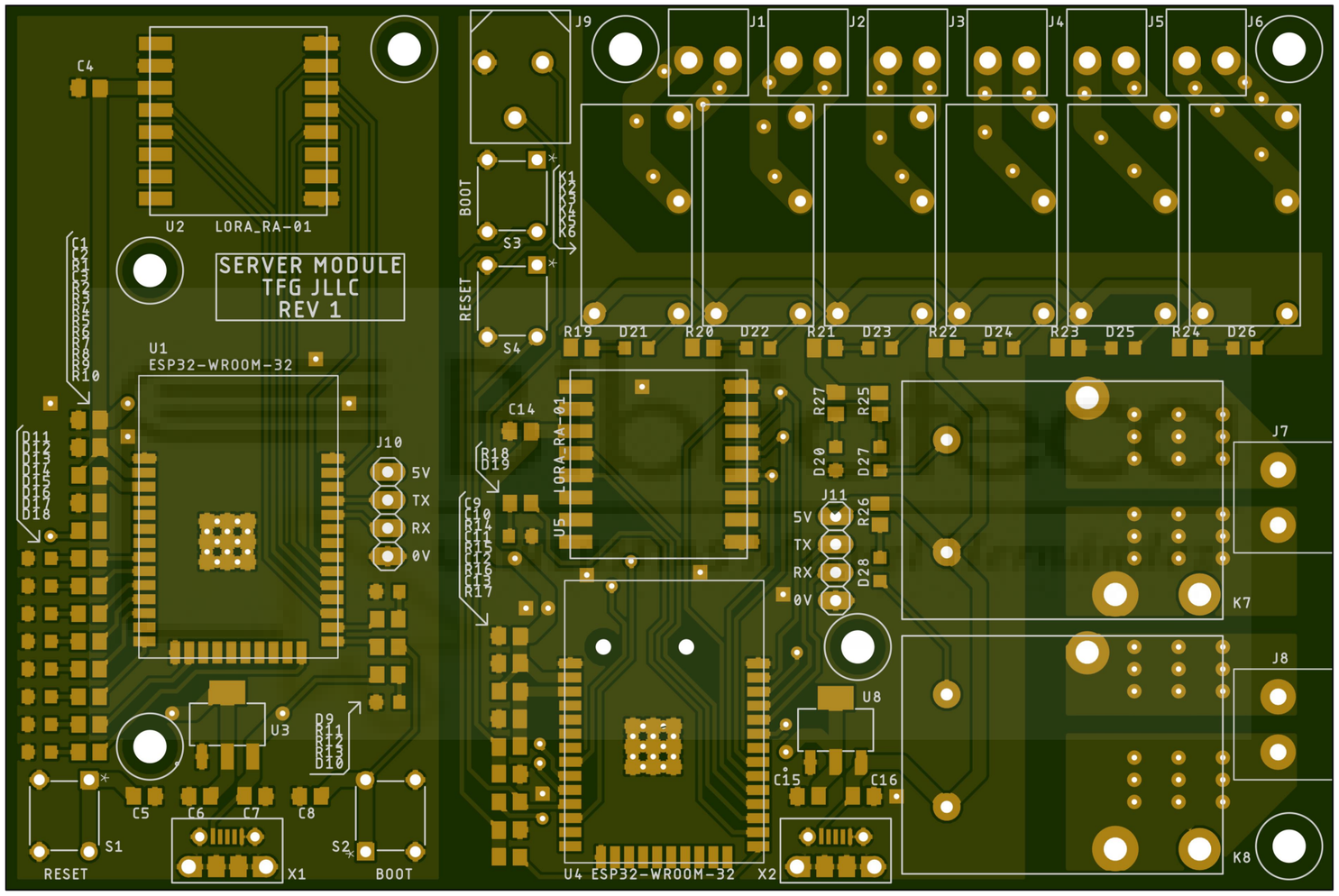




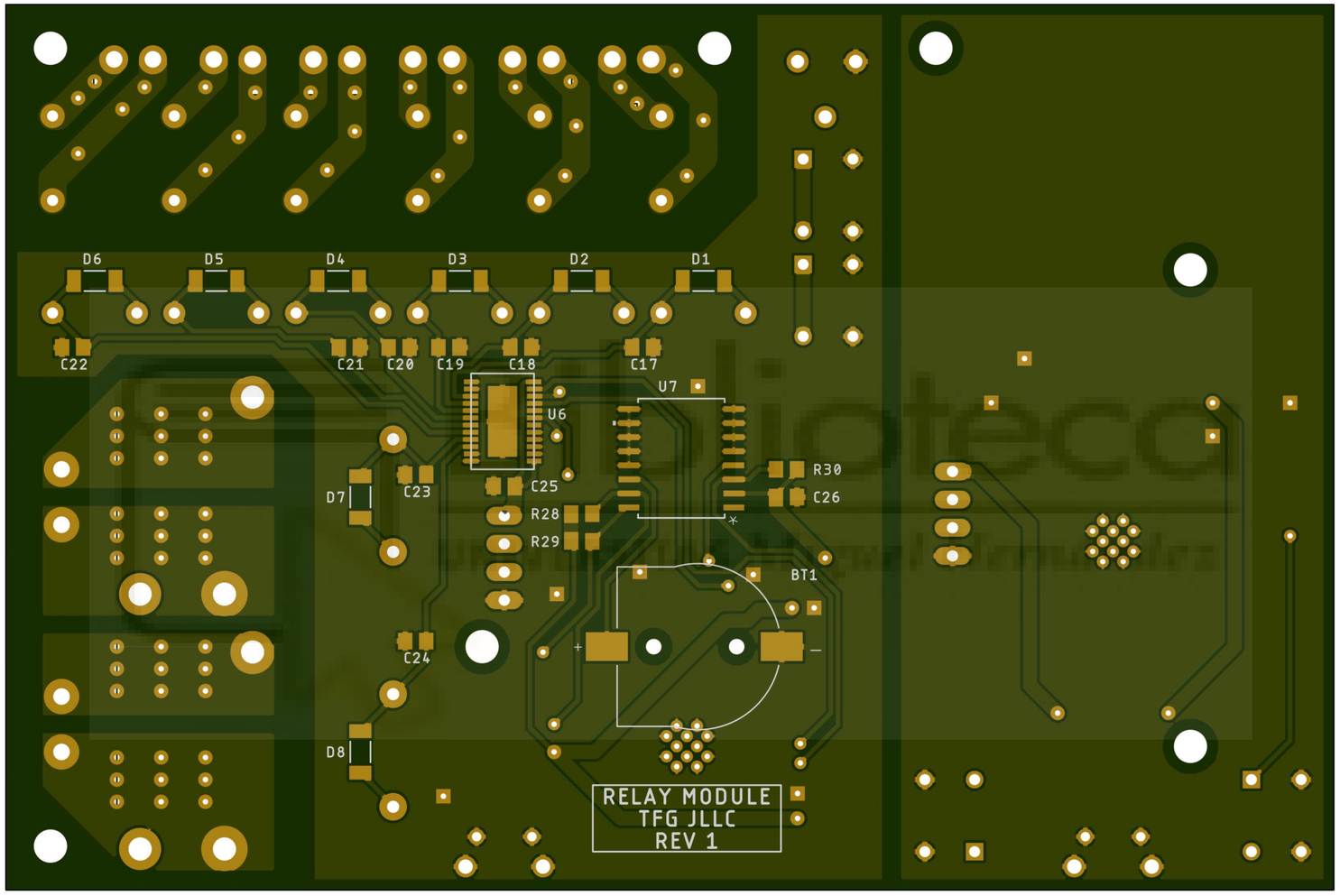




Escala	-	Proyectante	Juan Luis Leal Contreras	Fecha
TFG - PLC PARA CONTROL DE BOMBAS DE PISCINA Y SISTEMA DE RIEGO AUTOMÁTICO EN CHALÉS		Tutor	David Marroquí Sempere	22/09/2020
		ESQUEMÁTICO MÓDULO DE RELÉS		Nº Hoja 2 de 4



Escala	2:1	Proyectante	Juan Luis Leal Contreras	Fecha
TFG - PLC PARA CONTROL DE BOMBAS DE PISCINA Y SISTEMA DE RIEGO AUTOMÁTICO EN CHALÉS		Tutor	David Marroquí Sempere	22/09/2020
		PCB CARA SUPERIOR		Nº Hoja: 3 de 4



Escala	2:1	Proyectante	Juan Luis Leal Contreras	Fecha
TFG - PLC PARA CONTROL DE BOMBAS DE PISCINA Y SISTEMA DE RIEGO AUTOMÁTICO EN CHALÉS		Tutor	David Marroquí Sempere	22/09/2020
		PCB CARA INFERIOR		Nº Hoja 4 de 4