

POWER

Photovoltaic Optimized Wireless Energy Recharger

DAVID ANDRINO
ESTELA MORA
HUGO SÁNCHEZ
FERNANDO SANZ

Hardware para IoT
Otoño 2024

Índice

1. Introducción	1
1.1. Descripción funcional del equipo	1
1.2. Especificaciones funcionales	1
1.2.1. Especificaciones Hardware	1
1.2.2. Especificaciones Software	1
2. Desarrollo Hardware	3
2.1. Diagrama de bloques	3
2.2. Módulos individuales	3
2.2.1. ESP8266	3
2.2.2. Baterías	4
2.2.3. CN3768	4
2.2.4. INA226	5
2.2.5. Subsistema para los relés	6
2.2.6. XL6009	8
2.2.7. LM2596	9
2.2.8. AMS1117	10
2.2.9. PS58F1	11
2.3. Sistema completo	12
2.3.1. Conexionado del sistema	12
2.3.2. Estados del sistema	14
2.3.3. Esquemático del circuito completo	15
2.4. Montaje del prototipo	16
3. Software	20
3.1. Interfaz de Usuario	20
3.2. Módulos Software	24
3.2.1. Módulo manejador de ficheros	24
3.2.2. Fichero Hardware.h	26
3.2.3. Módulo INA	27
3.2.4. Módulo Cliente MQTT	29
3.2.5. Módulo Bajo Consumo	30
3.2.6. Módulo Telemetría	32
3.3. Programa Principal	32
3.4. Programa Lectura Fichero	37
3.5. Librerías Externas	39
3.5.1. Librería INA	39
3.5.2. Librería PubSubClient	39
3.6. Documentación del código	39
4. Consideraciones teóricas	41
4.1. Panel solar	41
4.2. Caja	43
4.2.1. Requisitos técnicos	43
4.2.2. Diseño 3D	43
4.2.3. Características técnicas	45
5. Presupuesto	46
6. Horas dedicadas al proyecto	47
7. Bibliografía	48
Anexo A. Código de la aplicación	49
Anexo B. Perspectivas de la caja	60

Índice de figuras

1.	Diagrama de bloques general de la solución	3
2.	ESP8266	3
3.	Conexiones ESP8266	4
4.	Módulo cargador CN3768	5
5.	INA226	5
6.	Possibles direcciones I ² C del INA226	6
7.	Esquemático básico del circuito de los relés	7
8.	PCB del circuito para los relés	8
9.	Respuesta subamortiguada del circuito del relé	8
10.	Módulo regulador elevador XL6009	9
11.	Módulo regulador reductor LM2596	9
12.	LDO AMS1117	10
13.	Componente alterna de la tensión de salida del regulador commutado	11
14.	Componente alterna de la tensión de salida del LDO	11
15.	Controlador de descarga PS58F1	12
16.	Diagrama de bloques simples del sistema	12
17.	Conexiones del sistema	13
18.	Sistema alimentado por el panel solar	14
19.	Sistema alimentado por la batería de backup	14
20.	Estado por defecto	15
21.	Esquemático del circuito completo	16
22.	PCB del circuito para los relés	16
23.	Conectores utilizados en el prototipado	17
24.	Placas para montaje de LDO y ESP8266	18
25.	Prototipo del circuito completo	18
26.	Plataforma ThingsBoard	20
27.	Redirección de puertos MV	21
28.	Tabla de medidas	22
29.	Gráfica de medidas	23
30.	Cadena de reglas para cálculo de potencias	23
31.	Cadena de reglas principal	23
32.	Página de gráficas ThingsBoard	24
33.	Página de tablas ThingsBoard	24
34.	SPIFFS	25
35.	Modos Bajo Consumo ESP8266	31
36.	Diagrama de flujo Programa Principal	35
37.	Mensaje inicial programa de lectura	37
38.	Captura de la web de documentación	40
39.	GitHub Action para la generación de documentación	40
40.	Prestaciones SOLARIMPUT PF200T	41
41.	Prestaciones LandStar LS1524	42
42.	Flujo de Corriente Panel Solar	42
43.	Vista Isométrica Modelo 3D caja	43
44.	Vista Puerta Usuario Modelo 3D caja	44
45.	Vista Puerta Técnico Modelo 3D caja	44
46.	Vista Incisiones Modelo 3D caja	44
47.	Funda cableado exterior	45
48.	Prensaestopa	45
49.	Vista isométrica	60
50.	Caja Cerrada	60
51.	Vista parte superior de la caja	61
52.	Vista parte inferior de la caja	61
53.	Vista parte derecha de la caja	62
54.	Vista parte izquierda de la caja	62
55.	Vista parte frontal de la caja	63

56.	Vista parte de atrás de la caja	63
-----	---------------------------------	----

Índice de fragmentos

1.	Funcion <code>Clear_file</code>	25
2.	Funcion <code>Read_file</code>	25
3.	Funcion <code>Write_file</code>	26
4.	Codigo función <code>setupINA226Sensors</code>	27
5.	Codigo función <code>reconfig_INAs</code>	27
6.	Codigo función <code>powerDownINA226</code>	28
7.	Codigo función <code>powerUpINA226</code>	28
8.	Codigo función <code>measureINA226</code>	28
9.	Instancia de las direcciones de los INA	29
10.	Función <code>reconnect</code>	29
11.	Funcion <code>publishTelemetry</code>	30
12.	Función bajo consumo	31
13.	Activación modo bajo consumo	31
14.	Restauración modo bajo consumo	32
15.	Estructura de datos para la telemetría	32
16.	Cabecera con los módulos y ficheros utilizados	32
17.	Definición de la macro de depuración	33
18.	Constantes de <code>MAIN_POWER</code>	33
19.	Codigo de la función <code>setup</code>	34
20.	Codigo de la función <code>loop</code>	34
21.	Definición función <code>setup_wifi</code>	36
22.	Desarrollo función <code>setup_wifi</code>	36
23.	Definición función <code>setRelays</code>	36
24.	Desarrollo función <code>setRelays</code>	37
25.	Programa lectura o eliminación de fichero	38
26.	Ejemplo de comentario Doxygen	39
27.	Fichero <code>file_management.hpp</code>	49
28.	Fichero <code>file_management.cpp</code>	49
29.	Fichero <code>hardware.hpp</code>	50
30.	Fichero <code>ina.cpp</code>	51
31.	Fichero <code>ina.hpp</code>	53
32.	Fichero <code>MAIN_POWER.ino</code>	54
33.	Fichero <code>mqtt_client.cpp</code>	56
34.	Fichero <code>mqtt_client.hpp</code>	57
35.	Fichero <code>sleep.cpp</code>	58
36.	Fichero <code>sleep.hpp</code>	58
37.	Fichero <code>telemetry.hpp</code>	58

Índice de tablas

1.	Conexiones del ESP8266	4
2.	Caracterización del rendimiento del XL6009	9
3.	Caracterización del rendimiento del LM2596	10
4.	Presupuesto del proyecto	46
5.	Distribución de horas dedicadas al proyecto	47

1. Introducción

1.1. Descripción funcional del equipo

El sistema se encarga de cargar mediante energía solar y monitorizar dos baterías 12V / 7Ah. Se encargará de guardar las medidas en un log y se visualizarán en un servidor, al que el usuario podrá acceder mediante un PC o un móvil. Además, el sistema dispondrá de una batería de backup para alimentar el sistema cuando la situación lo requiera (Bajo rendimiento solar). La batería de backup también se recargará mediante energía solar.

1.2. Especificaciones funcionales

Se diseñará un sistema de carga de baterías para emplazamientos aislados exteriores, que utilice energía solar como fuente de energía y que sea controlable y monitorizable a través de internet. El sistema deberá cumplir las siguientes especificaciones técnicas para asegurar su correcto y completo funcionamiento:

1.2.1. Especificaciones Hardware

- El sistema deberá cargar dos baterías a la vez (BAT1 y BAT2).
- Se utilizará un circuito específico para la carga de las baterías que debe contar con protección.
- El sistema priorizará el uso de la energía generada por el panel solar.
- El sistema contará con una tercera batería de backup, que se utilizará cuando el panel solar sea incapaz de entregar la energía necesaria.
- La batería de backup se cargará también mediante energía solar.
- Se utilizarán tres baterías de plomo ácido de 12V y 7Ah cada una.
- El panel solar debe ser capaz de generar suficiente energía para cargar ambas baterías en condiciones ideales.
- Se utilizará el microcontrolador ESP8266 para el control y monitoreo del sistema mediante I2C enviar las medidas por Wi-Fi.
- Se utilizará este sensor INA226 para medir la tensión y corriente de cada batería y el panel solar.
- El sistema debe ser resistente a condiciones climáticas adversas.
- El sistema deberá contar con un diseño que permita la adecuada ventilación y disipación del calor generado por la electrónica.
- Se minimizará al máximo el consumo del sistema encargado del control de las baterías.
- El sistema cumplirá las normativas de seguridad eléctrica y medioambiental.
- Se realizará pruebas de integración de todo el sistema.
- El sistema se conectará a una red Wi-Fi con acceso a un servidor.
- El sistema enviará las medidas a un servidor externo.

1.2.2. Especificaciones Software

- Se utilizará el entorno Arduino IDE para el desarrollo del código en el lenguaje Arduino.
- Se configurará y gestionará la conexión Wi-Fi en el ESP8266.
- Toda la información estará disponible al usuario mediante un servidor web y un log almacenado en la memoria flash del microcontrolador.

- En caso de pérdida de señal, el sistema seguirá almacenando medidas localmente.
- Se dispondrá de un modo de lectura de medidas para obtener el `log` de medidas a través de `Serial`.
- Se utilizará el protocolo de comunicación MQTT para la comunicación entre el ESP8266 y el servidor.
- El cliente podrá acceder al sistema a través de una interfaz web (en móvil u ordenador).
- En el servidor web se visualizará en tiempo real gráficas y valores que correspondan con el estado de carga de las baterías y la producción de energía del panel solar.
- En el `log` se almacenará la tensión y corriente de carga de las tres baterías y la producción de energía del panel solar.
- El funcionamiento normal del sistema consistirá en: El sistema se despierta, toma medidas, gestiona los relés, intenta conectarse a la red y enviar las medidas, almacena las medidas en memoria y se duerme hasta el siguiente ciclo. De esta forma se optimiza el consumo de batería.

2. Desarrollo Hardware

2.1. Diagrama de bloques

La solución propuesta consta de tres entidades, aunque dos pueden ser el mismo dispositivo:

- Sistema de carga y monitorización de baterías. Compuesto por la electrónica diseñada (Apartado 2.3) y la inteligencia del ESP8266.
- Servidor de medidas. Compuesto por un *Broker MQTT* al que enviar las medidas y un *Dashboard* en la que representarlas. En nuestro caso, se trata de una máquina virtual Ubuntu en la que se instala *ThingsBoard*, un software que cumple ambas funciones. (Apartado 3.1)
- Cliente web. Cualquier dispositivo con un navegador web puede acceder al *Dashboard* para visualizar la información.

Se puede ver el diagrama de bloques en la Figura 1

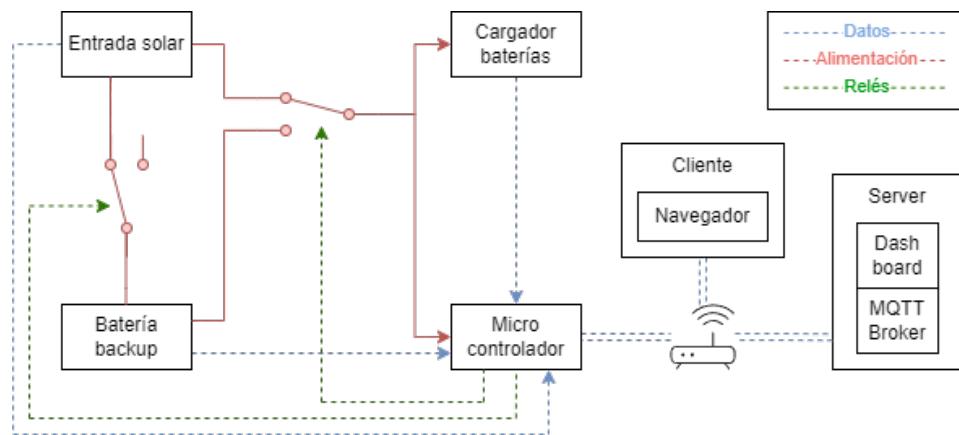


Figura 1: Diagrama de bloques general de la solución

2.2. Módulos individuales

2.2.1. ESP8266

El **ESP8266** es un microcontrolador de bajo coste que dispone de conexión Wi-Fi integrada. Para programarlo, se utiliza **Arduino IDE**, programando en lenguaje **Arduino**. [1]

En el sistema se utiliza para el monitoreo del sistema, gestión de medidas y control de los relés, así como para la conexión a la red Wi-Fi y el envío de las medidas a un servidor externo mediante el protocolo de comunicación MQTT.

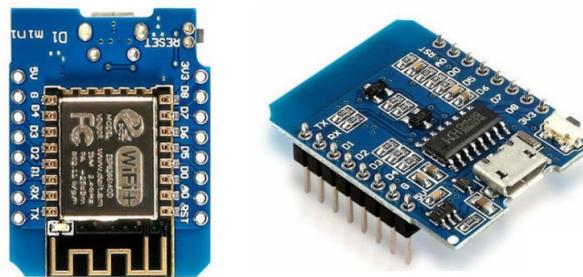


Figura 2: ESP8266

Se han utilizado los siguientes pines:

	Pin	GPIO
Relé para cargar la Backup	7	13
Relé para la alimentación	8	15
SDA	2	4
SCL	1	5
Alimentación ESP8266	5V	N/A
GND	GND	N/A

Tabla 1: Conexiones del ESP8266

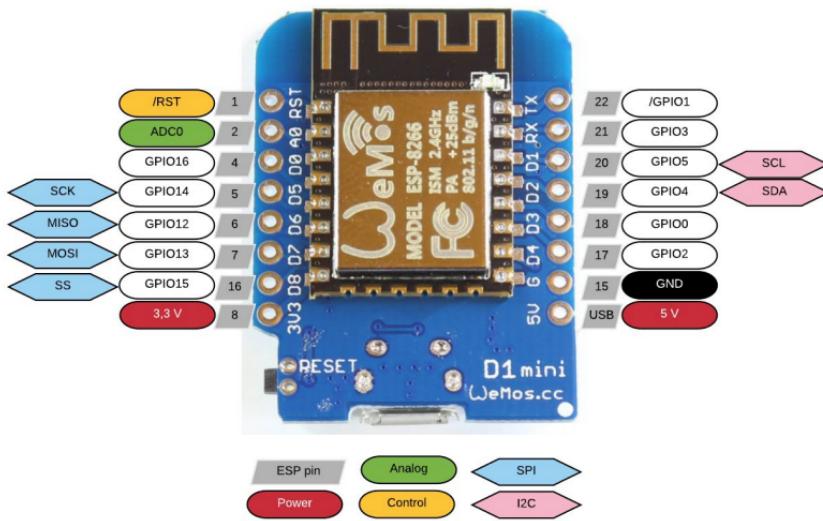


Figura 3: Conexiones ESP8266

2.2.2. Baterías

Para las baterías, se van a elegir unas baterías de plomo ácido. Estas tienen una tensión nominal de 12 V y una capacidad de 7 Ah. Estas son baterías típicamente utilizadas para herramientas de potencia como las que serían objetivo del caso de uso del sistema.

2.2.3. CN3768

Este módulo es el cargador de baterías, que utilizamos para cargar las tres. Se trata de un módulo con un controlador y un convertidor conmutado para generar la tensión de carga.

El fabricante indica que acepta tensiones de entre 15V y 24V, con una corriente de carga máxima de 4A, aunque la placa viene configurada para 200mA con una resistencia, lo cual hemos respetado. Indica que la tensión de carga es de 14,8V y que cuenta con una luz LED que debería indicar el estado de la carga. Sin embargo, hemos apreciado que el comportamiento de dicha luz es bastante poco constante, apagándose aleatoriamente durante el proceso de carga, aunque este siga en proceso. [2]

Un problema que no está indicado es la corriente en reversa. Este módulo cuenta con dos terminales de entrada y dos de salida. Cuando la tensión en los terminales de entrada es suficiente como para alimentar el módulo, se comporta adecuadamente y envía corriente a la batería para cargarla. Sin embargo, si la tensión en la entrada no es suficiente, el circuito permite corrientes en dirección contraria, descargando la batería. Esto lo hemos solucionado fácilmente con unos diodos 1N4007 en la entrada. Estos diodos cuentan con una corriente máxima de 1A y una caída aproximada de 0,6V. Esta caída de tensión no nos afecta, ya que trabajamos bastante por encima de la tensión mínima de entrada del cargador.



Figura 4: Módulo cargador CN3768

Experimentalmente, hemos comprobado que el cargador consume como máximo los 200 mA configurados durante la carga de la batería, aunque se reduce en algunos puntos de la carga.

2.2.4. INA226

El INA226 es un sensor de corriente y tensión que reporta las medidas a través de I₂C. En el sistema se utilizan para medir tanto la tensión como la corriente del panel solar, la batería de backup y las baterías a cargar (BAT1 y BAT2). [3]

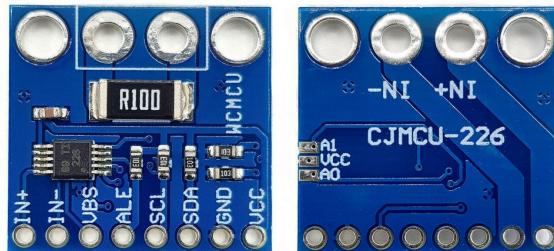


Figura 5: INA226

Para poder medir de forma precisa, la placa incluye un resistor **shunt**, una resistencia con valor muy bajo que se coloca en serie con la carga para medir la corriente que circula por ella. La tensión que cae en el resistor **shunt** es proporcional a la corriente que circula por él, y el INA226 mide esta tensión para calcular la corriente que circula por el circuito.

La dirección I₂C del sensor se puede configurar mediante los terminales A0 y A1, uniendo estos a VCC, GND, SDA o SCL. En nuestro caso, se ha configurado la dirección I₂C de la siguiente forma:

- INA226 del panel solar: 0x45
- INA226 de la batería de backup: 0x40
- INA226 de la batería BAT1: 0x44
- INA226 de la batería BAT2: 0x41

Table 6-2. Address Pins and Slave Addresses

A1	A0	SLAVE ADDRESS
GND	GND	1000000
GND	VS	1000001
GND	SDA	1000010
GND	SCL	1000011
VS	GND	1000100
VS	VS	1000101
VS	SDA	1000110
VS	SCL	1000111
SDA	GND	1001000
SDA	VS	1001001
SDA	SDA	1001010
SDA	SCL	1001011
SCL	GND	1001100
SCL	VS	1001101
SCL	SDA	1001110
SCL	SCL	1001111

Figura 6: Posibles direcciones I²C del INA226

Los INA226 tienen una elevada precisión de medida de tensión, pero la medida de corriente tiene una ligera imprecisión debido probablemente a la incertidumbre del valor de la resistencia de medida. Sin embargo, el integrado cuenta con un registro para compensar dicha imprecisión, aplicando un coeficiente que se puede ajustar experimentalmente.

2.2.5. Subsistema para los relés

La selección de tensión de alimentación para los cargadores y la carga o descarga de la batería de *Backup* se ha realizado mediante relés. Debido a la naturaleza del proyecto, es interesante reducir al mínimo posible el consumo de corriente de los elementos del circuito, por lo que el consumo constante de un relé no es algo admisible.

Se tuvo en consideración el uso de relés biestables, pero su elevado coste en comparación y la recomendación de realizar un subsistema electrónico con una PCB propia, decidimos tomar otra alternativa.

La corriente que un relé necesita para comutar es menor que la que necesita para mantener el interruptor conmutado, por lo que decidimos realizar un circuito que aplicara un pico de corriente al comutar y redujera la corriente para mantener la comutación.

Se puede ver el circuito en la Figura 7. Se utiliza un condensador y una resistencia que consiguen una respuesta subamortiguada ante el escalón de comutación. Además, se utiliza un transistor para manejar la comutación del relé y se utiliza un diodo de *flyback* para proteger a dicho transistor de la corriente de la bobina cuando se intente cortar. Para la resistencia en paralelo con el condensador se ha utilizado un potenciómetro para poder ajustar el pico de comutación.

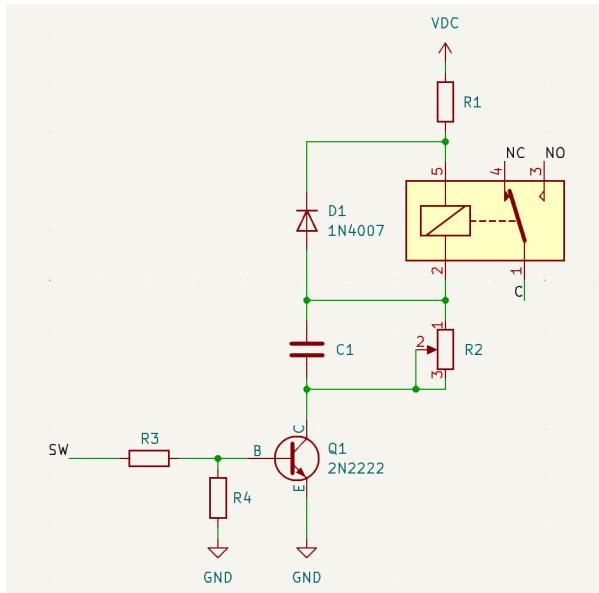
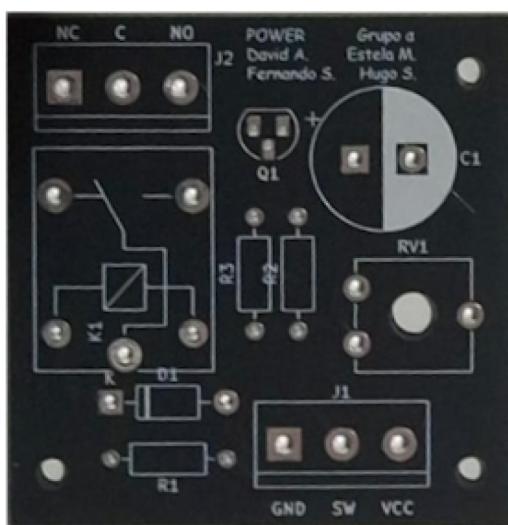


Figura 7: Esquemático básico del circuito de los relés

El valor de los componentes pasivos se ha elegido experimentalmente, en función de los valores que mejores resultados han dado en el laboratorio. El propósito de cada componente es:

- R1: Reduce la tensión aplicada en la bobina, ya que esta es de 12 V y este circuito se puede alimentar con 24 V. Se ha utilizado un valor de 100 Ω .
- R2: Potenciómetro que junto a C1 crean la respuesta subamortiguada en la conmutación. Se ha utilizado un valor de 2,2 k Ω .
- R3: Resistencia para limitar la corriente de base del transistor, ya que va a venir de un GPIO del microcontrolador. Se ha utilizado un valor de 100 Ω .
- R4: Resistencia de *pull-down* para colocar un nivel bajo débil en la base del transistor. Se ha utilizado un valor de 1 k Ω .
- C1: Condensador utilizado para la respuesta subamortiguada del sistema. Inicialmente, se encuentra descargado, por lo que actúa como un cortocircuito, creando el pico de corriente. Según se va cargando, va tomando importancia la resistencia y esta es la que decide la corriente final del relé. Se ha utilizado un valor de 1000 μF .
- Q1: Transistor encargado de la conmutación del relé. Es accionado desde un GPIO del microcontrolador. Se ha utilizado el transistor PNP modelo 2N2222.
- D1: Diodo de *flyback* utilizado para redirigir la corriente de la bobina cuando se corta el transistor, para evitar destruirlo. Se ha utilizado un diodo 1N4007 por ser el más accesible, aunque lo óptimo hubiera sido un diodo Schottky.

Para el montaje de dicho circuito se ha diseñado una PCB que se ha enviado a fabricar a una empresa extranjera. El resultado se puede ver en la Figura 8. Las resistencias soldadas en la imagen no tienen los valores indicados porque se cambiaron después de hacer la foto.



(a) PCB sin montar



(b) Circuito montado

Figura 8: PCB del circuito para los relés

Observando la forma de onda de la corriente de conmutación (proporcional a la tensión en la resistencia R_1), se ve la diseñada respuesta subamortiguada.

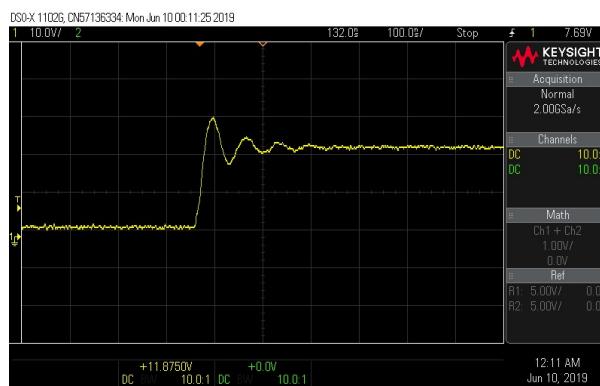


Figura 9: Respuesta subamortiguada del circuito del relé

Con este circuito alimentado a 24 V, se consigue reducir la corriente de un relé desde 60,2 mA a 14,1 mA.

2.2.6. XL6009

El regulador conmutado elevador (o *Boost converter* en inglés) es un módulo que permite elevar una tensión de entrada más baja a una tensión de salida mayor. Esto se logra gracias a un interruptor (transistor) controlado que conecta una bobina a la alimentación, almacenando mucha energía en forma de campo magnético y redirigiéndola en forma de corriente a través de un diodo al circuito, filtrando a través de un condensador. Una consecuencia es que incrementa la corriente requerida proporcionalmente a la relación de tensión elevada. [4]

En este caso, se utiliza para elevar la tensión de la batería de backup (tensión de 12V) a una tensión de 24V para alimentar el sistema, ocupando el lugar del panel cuando este no esté disponible.



Figura 10: Módulo regulador elevador XL6009

Con el fin de caracterizar el rendimiento, se han medido las tensiones y corrientes de entrada para dos valores de resistencia, calculando el rendimiento en ambos casos.

Carga	V_{in}	I_{in}	V_{out}	I_{out}
900 Ω	11,87722 V	83,476 mA	24,9900 V	27,415 mA
450 Ω	11,74912 V	148,190 mA	24,9781 V	54,196 mA

(a) Medidas tomadas en laboratorio

Carga	$P_{in} = V_{in} \cdot I_{in}$	$P_{out} = V_{out} \cdot I_{out}$	$\eta = 100 \% \cdot \frac{P_{out}}{P_{in}}$
900 Ω	0,991 W	0,685 W	69,1 %
100 Ω	1,741 W	1,353 W	77,75 %

(b) Cálculos de rendimiento realizados

Tabla 2: Caracterización del rendimiento del XL6009

Como se puede ver, el rendimiento incrementa con la corriente, como es natural en los reguladores conmutados, por lo que en nuestra aplicación se pueden esperar rendimientos incluso mejores.

2.2.7. LM2596

El regulador conmutado reductor (o *Buck converter* en inglés) es un módulo que permite reducir una tensión de entrada alta a una tensión de salida menor. Esto se realiza mediante un interruptor que comuta la corriente que atraviesa un filtro paso bajo LC. Además, cuenta con la virtud de que reduce la corriente de entrada, al contrario que los reguladores lineales. [5]

En este caso, se utiliza para reducir la tensión de alimentación (tensión de 24V) a una tensión de 7V para la entrada del LDO, que alimentará a los INA226 y al ESP8266.

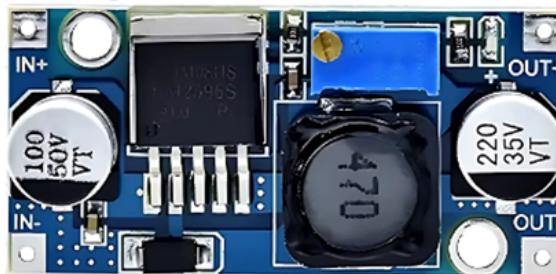


Figura 11: Módulo regulador reductor LM2596

Con el fin de caracterizar el rendimiento, se han medido las tensiones y corrientes de entrada para dos valores de resistencia, calculando el rendimiento en ambos casos.

Carga	V_{in}	I_{in}	V_{out}	I_{out}
900 Ω	12,0045 V	10,4268 mA	4,99920 V	5,4825 mA
100 Ω	11,9632 V	34,8312 mA	4,98207 V	47,7207 mA

(a) Medidas tomadas en laboratorio

Carga	$P_{in} = V_{in} \cdot I_{in}$	$P_{out} = V_{out} \cdot I_{out}$	$\eta = 100\% \cdot \frac{P_{out}}{P_{in}}$
900 Ω	125,168 mW	27,408 mW	21,897 %
100 Ω	416,693 mW	237,748 mW	57,056 %

(b) Cálculos de rendimiento realizados

Tabla 3: Caracterización del rendimiento del LM2596

Como se puede ver, el rendimiento incrementa con la corriente, como es natural en los reguladores conmutados, por lo que en nuestra aplicación se pueden esperar rendimientos incluso mejores.

2.2.8. AMS1117

El módulo AMS1117 es un regulador de voltaje de 5V, el cual se utiliza para alimentar el microcontrolador. Es un regulador lineal de bajo *drop-out* utilizado para suavizar la alimentación del microcontrolador.

En su hoja de características indica que (en nuestra versión) tiene una tensión de salida fija de 5V, un *drop-out* mínimo de 1,3V en el peor caso, una regulación de línea de hasta 10mV para nuestra tensión de entrada y salida diseñadas, regulación de carga de hasta 35mV y una corriente máxima de 0,9A [6].

Utilizaremos el regulador a la salida del regulador reductor, con la finalidad de suavizar los picos de tensión que aparecen por la naturaleza conmutada de dicho regulador.

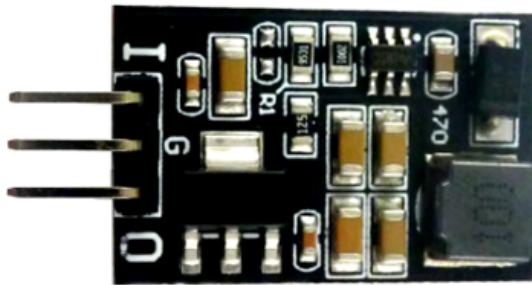


Figura 12: LDO AMS1117

En la Figura 13 se puede ver que la tensión de salida del regulador presenta picos de aproximadamente 110 mV, que se eliminan gracias a este componente. Aparece en la tensión de salida de este componente una oscilación de 20 mV, por lo que se ha reducido significativamente.

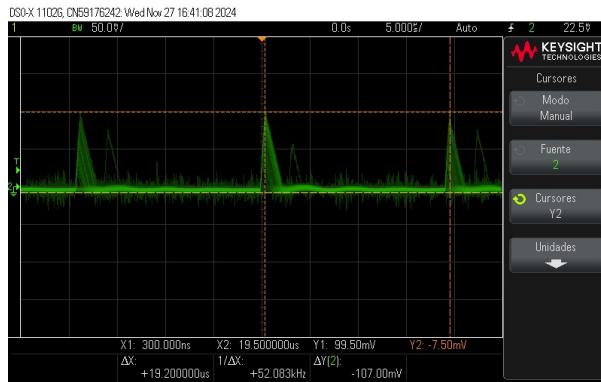


Figura 13: Componente alterna de la tensión de salida del regulador conmutado

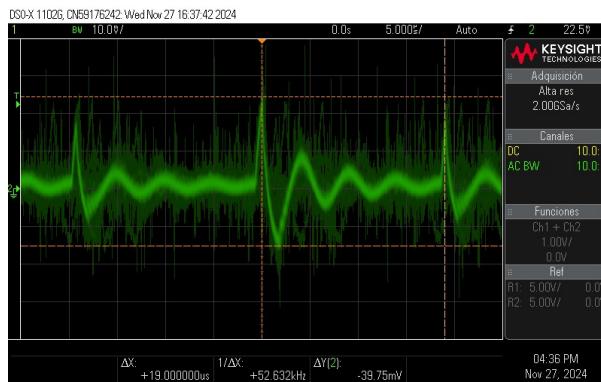


Figura 14: Componente alterna de la tensión de salida del LDO

Experimentalmente, hemos comprobado que la tensión de salida no converge hacia los 5 V anunciados para tensiones menores de 6,7 V, por lo que el *drop-out* es ligeramente superior al especificado. Sin embargo, como lo alimentamos con un regulador conmutado regulable no es ningún problema, solamente incrementa ligeramente la disipación de potencia del regulador.

En la caracterización, se han medido dos valores:

- Regulación de línea: Se han medido valores de tensión de salida variando la tensión de entrada, obteniendo el siguiente valor:

$$\text{Reg. Línea} \equiv \frac{\Delta V_o}{\Delta V_i} = \frac{V_{o2} - V_{o1}}{V_{i2} - V_{i1}} = \frac{5,00376 - 5,00372}{8 - 7,2} = 50 \mu V/V$$

- Regulación de carga: Se han aplicado distintas resistencias a la salida del regulador y se ha medido la variación de la tensión de salida en función de dicha carga:

$$\text{Reg. Carga} \equiv \frac{\Delta V_o}{\Delta I_o} = \frac{V_{o2} - V_{o1}}{I_{o2} - I_{o1}} = \frac{V_{o2} - V_{o1}}{\frac{V_{o2}}{R_2} - \frac{V_{o1}}{R_1}} = \frac{5,00345 - 5,00262}{5,00345/900 - 5,00262/100} = -18,67 mV/A$$

2.2.9. PS58F1

El módulo PS58F1 es un controlador de tensión con un relé que se puede utilizar para muchas aplicaciones. En nuestro caso, nos interesaba la aplicación como controlador de descarga para la batería de plomo ácido, de manera que desconectara la batería cuando esta sobrepasara un umbral.

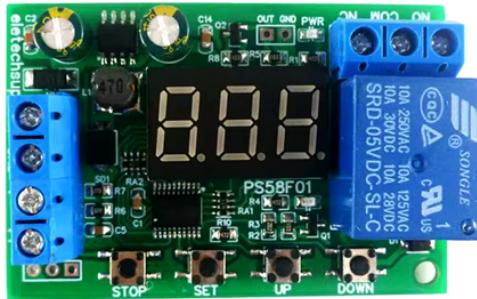


Figura 15: Controlador de descarga PS58F1

Este módulo cuenta con un relé que es el que se utiliza para la conexión o desconexión de la batería. Sin embargo, este módulo cuenta con muy poca documentación. Además, tras conseguir configurarlo para que aproximadamente funcione, hemos apreciado que cuenta con una muy baja precisión y la utilización de un relé hace que cuente con un consumo no despreciable. Como punto final, la comutación es inestable y a veces oscila en lugar de realizar una desconexión permanente.

Por todos estos factores, hemos decidido desechar este módulo y realizar un control de la descarga por software. Este control se explicará más profundamente en el Subapartado 2.3.1.

2.3. Sistema completo

El circuito electrónico construido para el sistema está resumido en la Figura 16. En ella se puede ver que se tiene la entrada solar, la cual puede cargar la batería de *Backup* y un seleccionador de entrada para el sistema, que puede alimentarse de la tensión solar o de la batería de *Backup*. De esta entrada se alimentan los dos cargadores de las baterías y el microcontrolador. Además, el módulo del microcontrolador alimenta los sensores de tensión y corriente, se comunica con el bus I2C e interactúa con los relés.

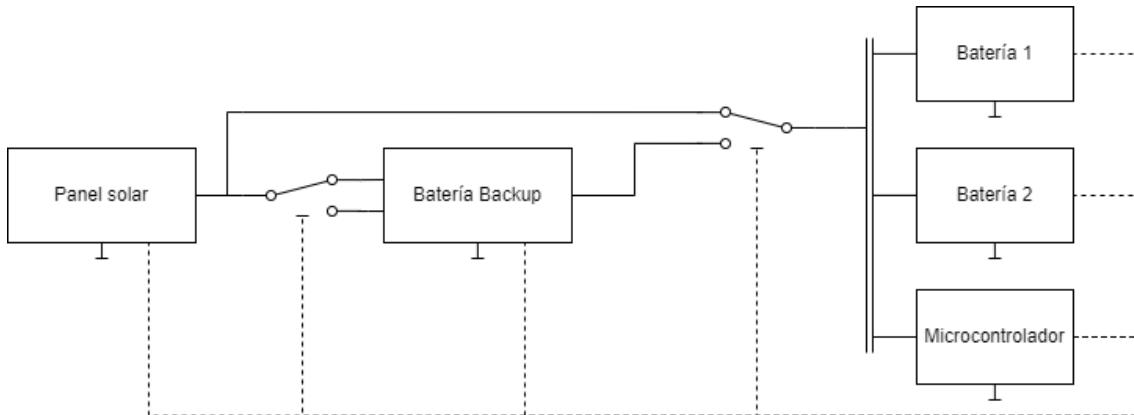


Figura 16: Diagrama de bloques simples del sistema

2.3.1. Conexionado del sistema

El sistema funciona principalmente gracias al panel solar. Este se encarga de alimentar el circuito y cargar las 3 baterías (BACKUP, BAT1 y BAT2). Para ello, ofrece una tensión de entorno a los 24V. Si se detecta que la tensión obtenida del panel es inferior a cierto umbral definido en el código, el circuito pasa a alimentarse con la batería de backup.

El panel solar se conecta a un INA226 para poder medir su tensión y corriente, y después se conecta a los dos relés. El relé 1 se encarga de conectar el panel solar al cargador de la batería de backup para poder cargarla, mientras que el relé 2 se encarga de elegir la alimentación del circuito (Solar o backup).

Entre el relé 1 y el cargador de la batería de backup hay un diodo. Este diodo evita la corriente inversa del cargador. Cabe destacar que este diodo no se ha colocado en la batería dado que evitaría corriente saliente de la batería, por lo que no permitiría utilizar la batería de backup como alimentación.

El cargador de la batería de backup se conecta a la batería de backup y a los INA226 para poder medir la tensión y corriente de la batería de backup. La batería de backup, para que pueda alimentar al circuito, se conecta a un regulador conmutado elevador, para convertir los 12V que entrega a 24V. Este regulador se conecta al relé 2 para poder alimentar el circuito.

Del relé 2 sale la alimentación del circuito, solar o backup según como esté comutado el relé. Esta alimentación se conecta a los cargadores de las baterías 1 y 2 y a un regulador conmutado reductor. Los cargadores de las baterías 1 y 2 se conectan a las baterías 1 y 2 respectivamente con los INA226 para poder medir la tensión y corriente de las baterías.

Entre la salida del relé 2 y cada cargador de batería hay un diodo. Este diodo, al igual que el de la de backup, evita la corriente inversa por el cargador. Cabe destacar que no se conecta entre el cargador y el INA226, ya que a la tensión para cargar la batería se le restaría la caída de tensión en el diodo, por lo que no se cargaría correctamente y para los 24V que entrega la alimentación sí se puede permitir la caída de tensión del diodo. Del mismo modo, tampoco se puede colocar entre el INA226 y la batería dado que no permitiría medir correctamente la tensión y corriente de carga de la batería, además de que seguiría existiendo el problema con la caída de tensión.

El regulador conmutado reductor se encarga de reducir los 24V de su entrada a 7V en la entrada del LDO. La salida del LDO se conecta a los INA226 y al ESP8266 para alimentarlos a 5V.

Para medir la tensión y corriente en el panel solar, la batería de backup y las baterías principales, se utilizan los INA226. Estos sensores se conectan al ESP8266 por I2C, que se encarga de procesar la información y enviarla por MQTT al servidor, además de guardarla en un log.

El ESP8266 se encarga no solo de la toma y gestión de las medidas sino que también del control de la conmutación de los relés para los cambios de estado del sistema [Subapartado 2.3.2] y la protección del circuito. Por ello, dos de sus pines están conectados a la entrada SW de cada relé.

En la Figura 17 se muestra el diagrama de conexiones del sistema:

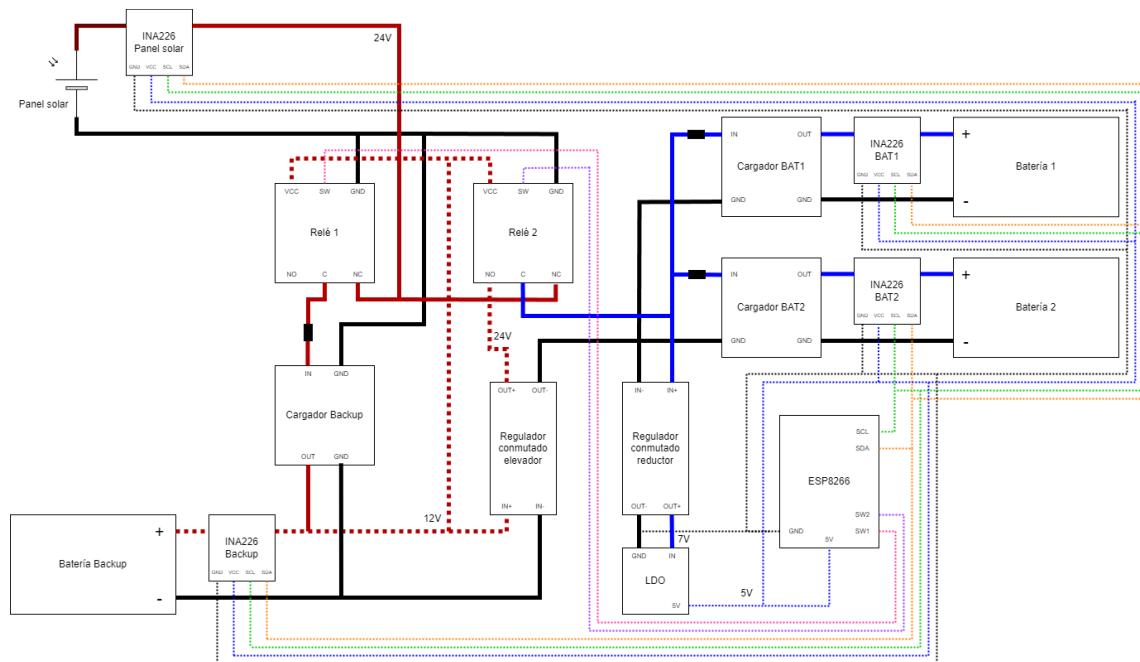


Figura 17: Conexiones del sistema

2.3.2. Estados del sistema

Los estados del sistema se controlan midiendo la tensión del panel solar y de la batería de backup con los INA226 y cambiando los relés con el ESP8266. En las figuras se han omitido los componentes que no son relevantes para la explicación de los estados del sistema, es decir, los convertidores, cargadores e INA226. Ahora se procede a explicar cada posible estado del sistema.

Por defecto, el sistema iniciará **alimentado por el panel solar**. En este estado, el sistema se alimenta directamente del panel solar y carga las baterías BACKUP, BAT1 y BAT2.

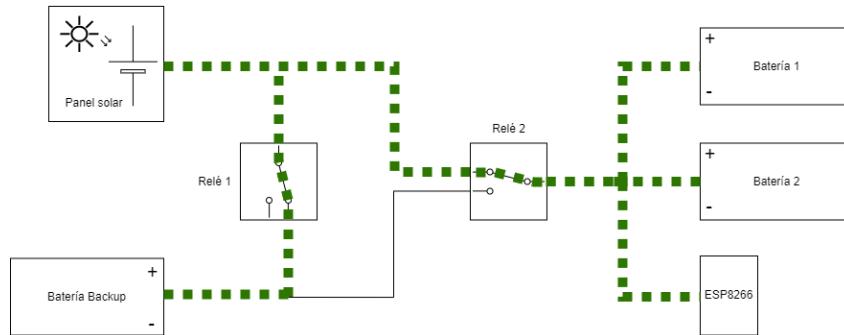


Figura 18: Sistema alimentado por el panel solar

En caso de que el panel solar no proporcione suficiente energía, el sistema pasará a **alimentarse de la batería de backup** gracias a los relés. En este estado, el sistema se alimenta de la batería de backup y carga las baterías BAT1 y BAT2. Cuando se detecte que el panel solar vuelve a proporcionar suficiente energía, el sistema volverá al estado anterior.

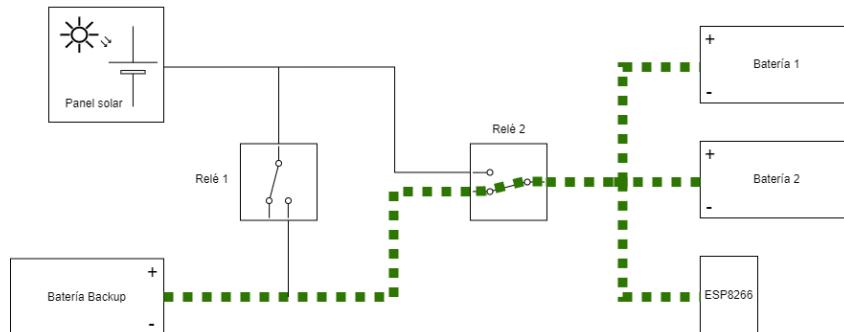


Figura 19: Sistema alimentado por la batería de backup

En caso de que la tensión de la batería alcance el umbral de sobredescarga, se pasará al **estado de protección**. Este estado tiene los relés en la misma posición que el **estado alimentado por panel solar** solo que el panel solar no es capaz de alimentar al circuito. Por tanto, el circuito no estará en funcionamiento, esperando a una tensión solar suficiente que permita el funcionamiento normal del sistema, pasando al **estado alimentado por panel solar**.

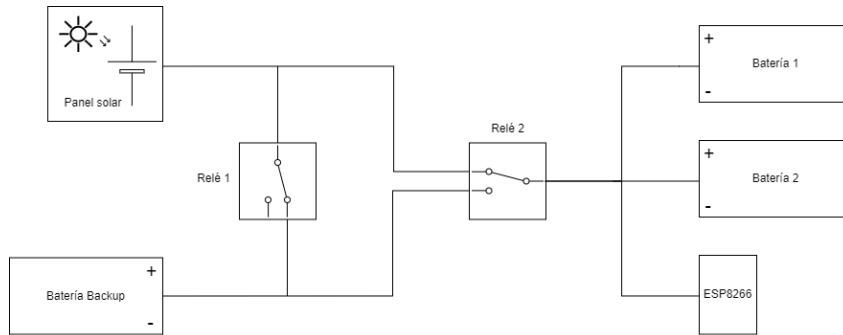


Figura 20: Estado por defecto

2.3.3. Esquemático del circuito completo

Se puede ver un esquemático completo de la solución en la Figura 21. Se pueden ver los módulos con las conexiones entre ellos. En él, se pueden ver todas las conexiones explicadas previamente. Está dividido en las seis zonas principales del circuito:

- Entrada solar. Compuesta por el panel solar simulado y su módulo de monitorizado de tensión y corriente. Se encarga de proporcionar la tensión de entrada al circuito (cuando no se está utilizando la batería de *backup*) y monitorizar sus parámetros.
- Batería de backup. Compuesta por el relé que conecta o desconecta la carga de la batería de backup, el cargador, el módulo de monitorización de tensión y corriente, la propia batería y el convertidor elevador. Se encarga de cargar la batería de *backup*, monitorizarla y elevar su tensión para utilizarla de entrada al circuito.
- Selección de fuente. Módulo de relé que commuta entre la tensión del panel y la de la batería de *backup* para utilizarlas como entrada al circuito.
- Baterías principales. Compuesta por los dos cargadores y módulos de monitorización de las baterías a cargar.
- Regulador 5 V. Regulador reductor conmutado y regulador lineal para conseguir la tensión de alimentación para el ESP8266 y los cuatro INA226.
- Microcontrolador. ESP8266 encargado de tomar las medidas, almacenarlas y enviarlas y gestionar la commutación de los relés.

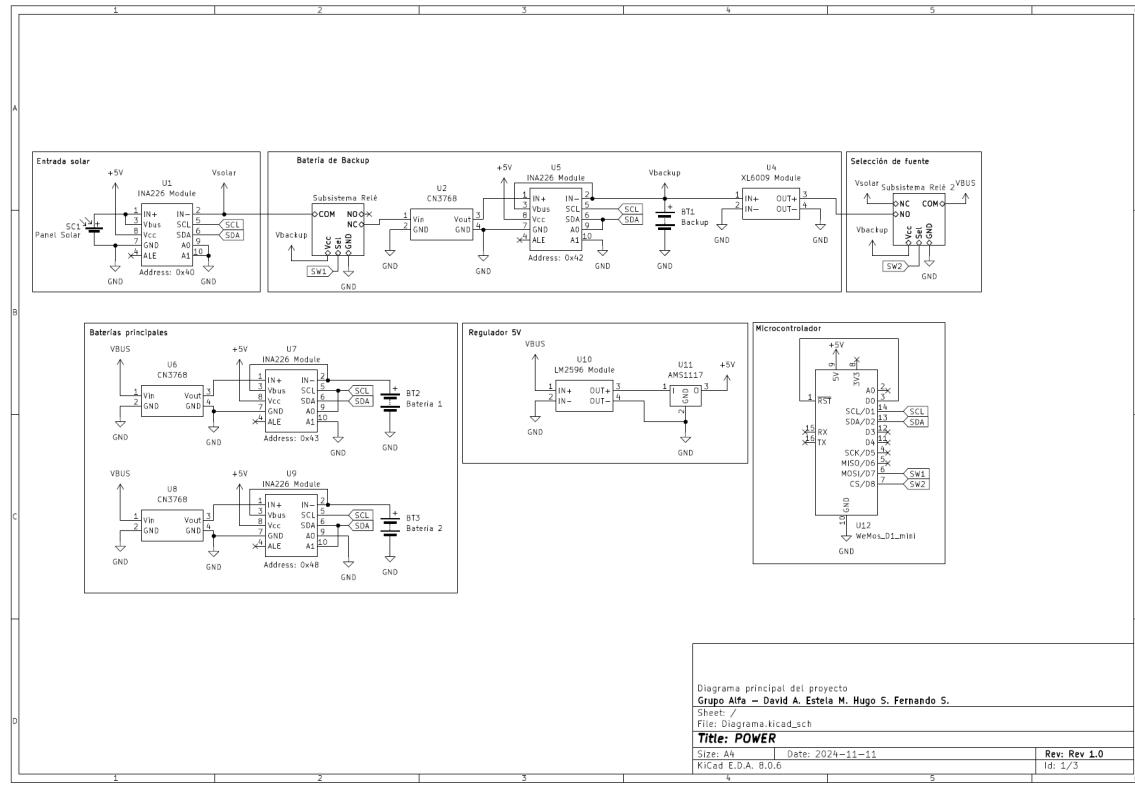
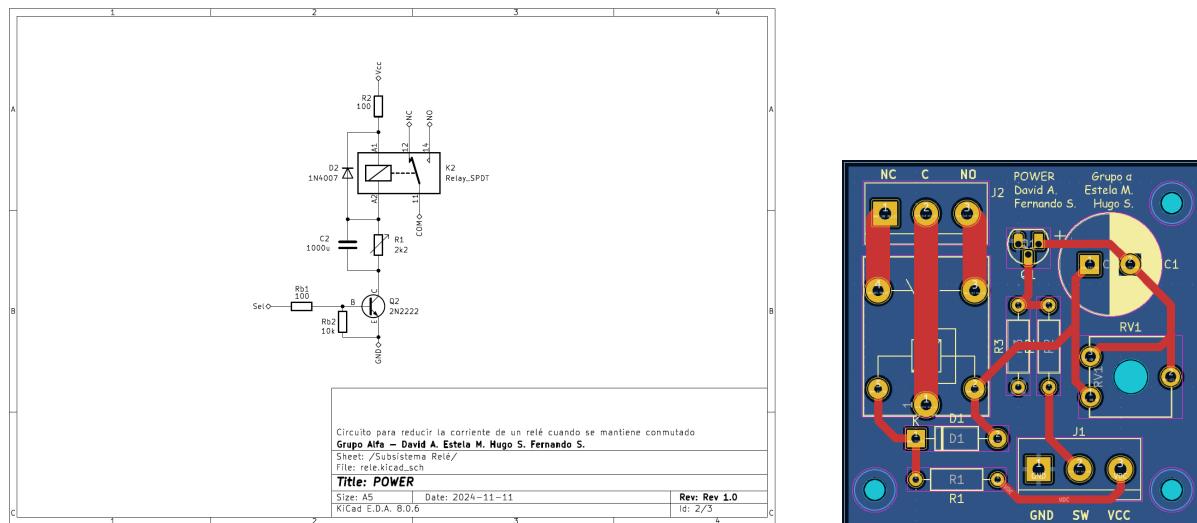


Figura 21: Esquemático del circuito completo

En la Figura 22 se puede ver el esquemático preciso del circuito para los relés y el diseño de la PCB construida. La explicación de este circuito se puede ver en el Subapartado 2.2.5.



(a) Esquemático del circuito para los relés

(b) Circuito montado

Figura 22: PCB del circuito para los relés

2.4. Montaje del prototipo

Para hacer el montaje del sistema en el prototipado, se ha utilizado una placa de madera de contrachapado para poder atornillar los componentes y que queden fijos. Se ha podido aprovechar que casi todas las placas de los módulos cuentan con perforaciones que hemos podido utilizar para colocar tornillos.

Las conexiones entre cables, para asegurar su firmeza y seguridad sin comprometer la rápida conexión y modificación del circuito, han sido realizadas con regletas de clemas y con conectores de conexión rápida. Por otro lado, las conexiones con algunos módulos se han podido realizar con terminales de tornillo. Por último, hemos utilizado abrazaderas de clavo para guiar los cables y fijar las dos placas que no tienen agujeros. Todos estos conectores se pueden ver en la Figura 23.



(a) Conexión rápida



(b) Regletas de clemas



(c) Abrazaderas de clavo



(d) Conejito de pala

Figura 23: Conectores utilizados en el prototipado

Tanto el LDO como el ESP8266 no cuentan con tornillos para su montaje, por lo que se ha realizado una placa adaptadora de baquelita. Además, se ha aprovechado para realizar las conexiones de los pines del ESP8266 en dicha placa, facilitando la conexión con el circuito. Se pueden ver las placas en la Figura 24.

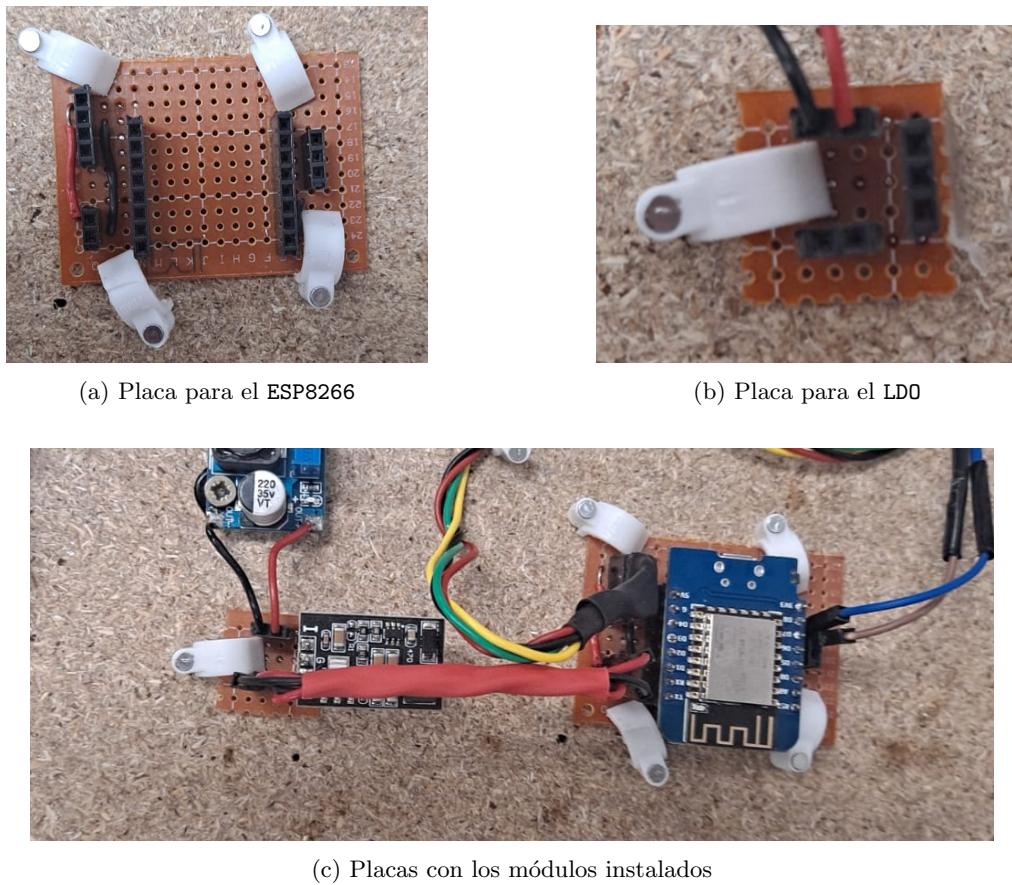


Figura 24: Placas para montaje de LDO y ESP8266

Se puede ver el prototipo final en la Figura 25.

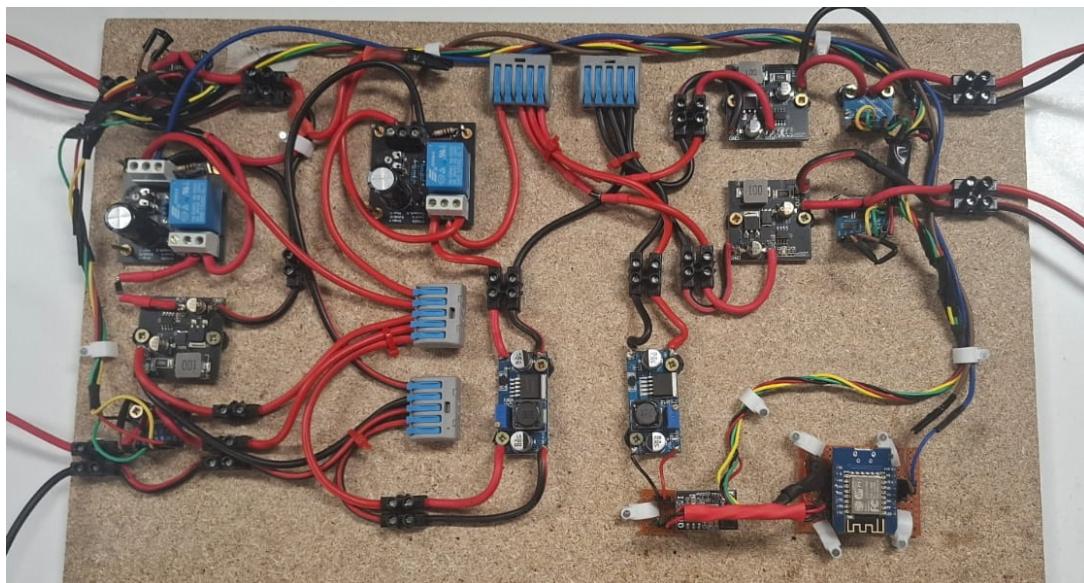


Figura 25: Prototipo del circuito completo

Con el fin de facilitar el prototipado, se utiliza la fuente de tensión del laboratorio para simular el panel solar. Cabe destacar que **la fuente no puede desconectarse por completo inmediatamente**, hay que disminuir la tensión hasta que se active la batería de *Backup* y entonces se puede desactivar. Esto

no es un problema en un sistema real, ya que la tensión de un panel solar no cae a 0 inmediatamente sino que disminuye progresivamente.

Además, la batería de *Backup* se puede simular con otra fuente de tensión **siempre que se desconecte la salida del cargador de dicha batería**, facilitado por el conector rápido al que está conectado.

3. Software

3.1. Interfaz de Usuario

Para la visualización y representación de los distintos valores de tensión y corriente medidos por los diferentes sensores, se ha optado por utilizar **ThingsBoard**, una plataforma IoT de código abierto para la recopilación, el procesamiento, la visualización y la gestión de dispositivos de datos.

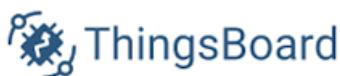


Figura 26: Plataforma ThingsBoard

Para su instalación en el ordenador, se ha optado utilizar una máquina virtual en VirtualBox mediante la imagen de un servidor Linux con distribución Ubuntu. Para la realización de dicha instalación se ha seguido la guía oficial de la plataforma de **ThingsBoard**, que se puede definir en los siguientes pasos: [7]

1. Instalar Java17 y configurarlo como predeterminado mediante los comandos:

```
sudo apt update
sudo apt install openjdk-17-jdk
sudo update-alternatives --config java
```

2. Instalar ThingsBoard mediante los siguientes comandos:

```
wget https://github.com/thingsboard/thingsboard/releases/\
download/v3.8.1/thingsboard-3.8.1.deb
sudo dpkg -i thingsboard-3.8.1.deb
```

3. Configurar la base de datos de ThingsBoard:

```
sudo apt install -y postgresql-common
sudo /usr/share/postgresql-common/pgdg/apt.postgresql.org.sh
sudo apt -y install postgresql-16
sudo service postgresql start
sudo su - postgres
psql
```

A continuación, elige tu contraseña del **postgres** mediante el siguiente comando:

```
\password
```

Después pulsa “Ctrl + D” para volver atrás y utiliza el siguiente comando para conectarte a la base de datos **postgres**:

```
psql -U postgres -d postgres -h 127.0.0.1 -W
```

Ahora, crea la base de datos para ThingsBoard mediante el comando:

```
CREATE DATABASE thingsboard;
```

Por último, pulsa dos veces “Ctrl + D” para salir del PostgreSQL.

4. Modificar el archivo de configuración de ThingsBoard mediante el siguiente comando:

```
sudo nano /etc/thingsboard/conf/thingsboard.conf
```

A continuación, añade estas líneas al final del archivo y no olvides cambiar

`PUT_YOUR_POSTGRESQL_PASSWORD_HERE` por la contraseña del `postgres` que pusiste anteriormente:

```
# DB Configuration
export DATABASE_TS_TYPE=mysql
export SPRING_DATASOURCE_URL=jdbc:postgresql://localhost:5432/thingsboard
export SPRING_DATASOURCE_USERNAME=postgres
export SPRING_DATASOURCE_PASSWORD=PUT_YOUR_POSTGRESQL_PASSWORD_HERE
# Specify partitioning size for timestamp key-value storage.
# Allowed values: DAYS, MONTHS, YEARS, INDEFINITE.
export SQL_POSTGRES_TS_KV_PARTITIONING=MONTHS
```

5. Elegir el tipo de servicio de cola de ThingsBoard. En este caso se elige el caso por defecto, en memoria, por lo que no es necesario ningún paso adicional.

6. Ejecutar el script de instalación mediante el siguiente comando:

```
sudo /usr/share/thingsboard/bin/install/install.sh --loadDemo
```

7. Arrancar el servicio de ThingsBoard mediante el siguiente comando:

```
sudo service thingsboard start
```

Una vez realizados estos pasos, se podrá acceder a ThingsBoard en la siguiente dirección:

`http://<IP VM>:8080/`

Por defecto, ThingsBoard incluye tres diferentes perfiles para acceder a la plataforma:

- *System Administrator*: `sysadmin@thingsboard.org` / `sysadmin`
- *Tenant Administrator*: `tenant@thingsboard.org` / `tenant`
- *Customer User*: `customer@thingsboard.org` / `customer`

Debido al uso de la máquina virtual, se ha tenido que realizar una redirección de los puertos mediante la interfaz de red de la propia máquina, obteniendo la siguiente configuración:



Figura 27: Redirección de puertos MV

Por lo que, en nuestro caso, el puerto 1883 del MQTT Broker se corresponde con el puerto 4444 del anfitrión (*host*), el puerto 3333 del anfitrión se corresponde con el puerto 8080 de la máquina virtual y también se ha redirigido el puerto 22 de SSH al puerto 2222 de la máquina anfitriona.

Para la comunicación entre la plataforma ThingsBoard y el dispositivo ESP8266, utilizamos un Broker MQTT OpenSource ampliamente utilizado debido a su ligereza, lo que permite fácilmente emplearlo en gran número de ambientes, incluso si estos son de pocos recursos. A continuación se indica el comando necesario para su instalación, obtenido de la guía oficial de ThingsBoard: [8]

```
sudo apt-get install mosquitto-clients
```

Para comprobar que todo se ha realizado correctamente, se puede utilizar el siguiente comando:

```
mosquitto_pub -d -q 1 -h "$THINGSBOARD_HOST_NAME" -p "1883"  
-t "v1/devices/me/telemetry" -u "$ACCESS_TOKEN" -m {"ATTRIBUTE":25}
```

Donde los siguientes parámetros corresponden con:

- THINGSBOARD_HOST_NAME: dirección IP del servidor ThingsBoard, por ejemplo, localhost o 127.0.0.1.
- ACCESS_TOKEN: token de acceso único proporcionado por ThingsBoard para cada dispositivo.
- ATTRIBUTE: atributo asociado a dicho dispositivo, por ejemplo, temperature.

En caso de que la conexión y publicación se haya realizado de manera correcta, se obtendrá la siguiente respuesta:

```
Client mosqpub|xxx sending CONNECT  
Client mosqpub|xxx received CONNACK  
Client mosqpub|xxx sending PUBLISH  
(d0, q1, r0, m1, 'v1/devices/me/telemetry', ... (16 bytes))  
Client mosqpub|xxx received PUBACK (Mid: 1)  
Client mosqpub|xxx sending DISCONNECT
```

Una vez que se ha comprobado el correcto funcionamiento de la conexión entre la plataforma ThingsBoard y el broker y cliente MQTT, se puede continuar con la personalización de dicha plataforma.

Para la visualización de los datos recibidos, se ha optado por diseñar dos paneles o *Dashboards*, uno en representación en forma de tablas y otro en forma de gráficas. Ambos paneles se actualizan en tiempo real y cuentan con una tabla o gráfica para cada dispositivo, obtenido un total de 4 tablas y 4 gráficas. Además, en las gráficas se puede visualizar también la media de los últimos datos medidos.

Tensión	Corriente	Potencia
21.720 V	360.110 mA	7.822 W

Figura 28: Tabla de medidas

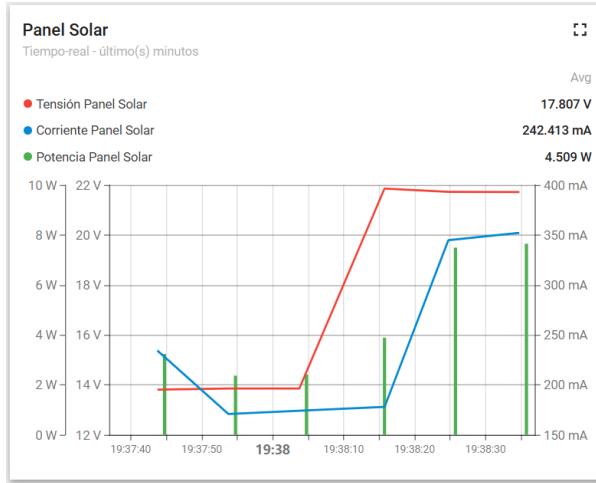


Figura 29: Gráfica de medidas

Debido a que mediante los sensores solo se obtienen valores de tensión y de corriente, se ha implementado un algoritmo mediante las cadenas de reglas de **ThingsBoard**. Se ha necesitado crear una cadena de reglas para cada potencia calculada, obteniendo 4 cadenas de reglas como la siguiente:

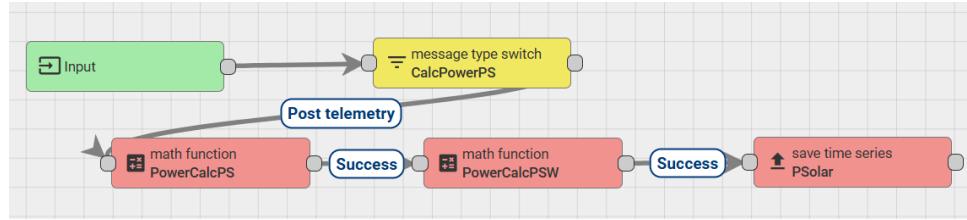


Figura 30: Cadena de reglas para cálculo de potencias

Dichas cadenas de reglas se dividen en los siguientes pasos:

1. Filtrar y transformar la telemetría entrante
2. Realizar el cálculo de la potencia correspondiente
3. Pasar dicha potencia a Vatios
4. Guardar los valores para su representación

Por último, se ha añadido dichas cadenas de reglas de potencia a la cadena de regla principal, la cual gestiona el funcionamiento completo de **ThingsBoard** y permite visualizar el dato obtenido de su respectiva cadena de regla.

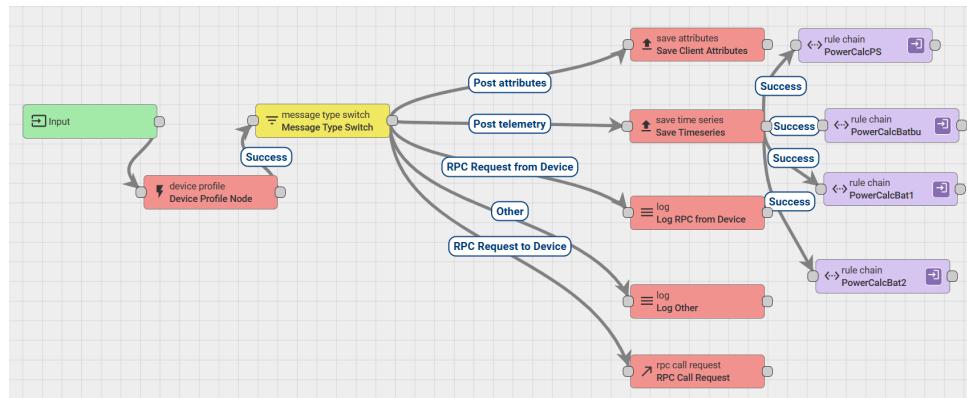


Figura 31: Cadena de reglas principal

Después de realizar los pasos descritos anteriormente, se han añadido las gráficas correspondientes a todas las medidas obtenidas en la página de gráficas del ThingsBoard, al igual que sus correspondientes tablas de valores instantáneos en la página de tablas, obteniendo las siguientes interfaces:

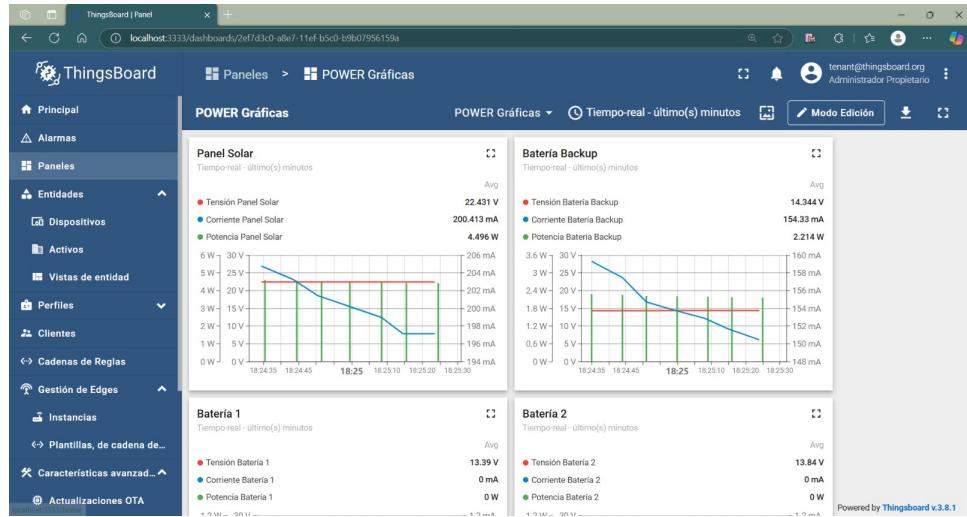


Figura 32: Página de gráficas ThingsBoard

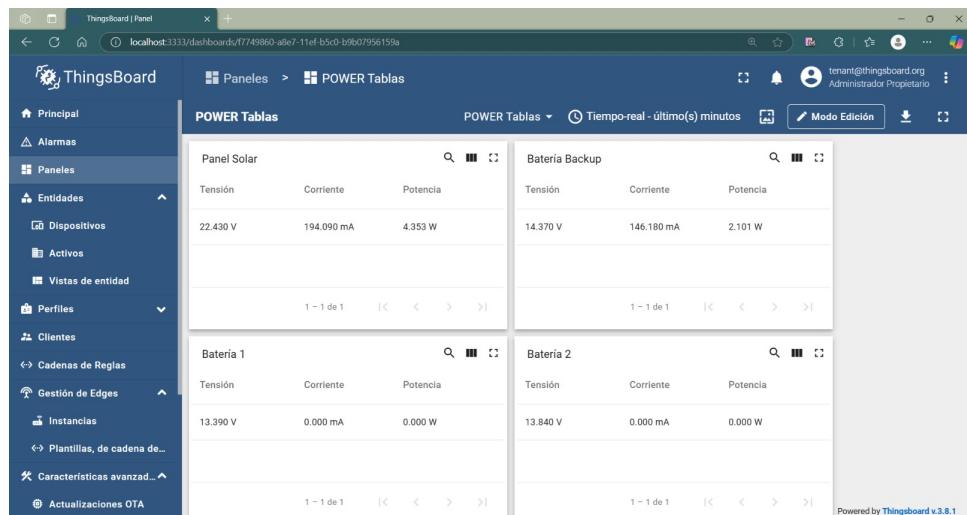


Figura 33: Página de tablas ThingsBoard

De esta forma, se obtiene una interfaz para visualizar los valores instantáneos en las tablas de las medidas obtenidas por los sensores a la par que se puede visualizar los valores históricos en las gráficas.

3.2. Módulos Software

3.2.1. Módulo manejador de ficheros

El módulo de gestión de fichero es el encargado de realizar tanto la escritura, lectura o eliminación del fichero que almacena las medidas obtenidas. Está formado por dos archivos, `file_management.cpp` el cual contiene las funciones necesarias, y `file_management.hpp`, el cual contiene la definición de las funciones y la importación de la librería externa utilizada.

Para la implementación de este módulo se ha requerido la utilización de SPIFFS incluido en la librería externa FS. SPIFFS o SPI Flash File System es un sistema de archivos diseñado para funcionar en memorias flash conectadas por SPI, lo que lo hace perfecto para este proyecto.



Figura 34: SPIFFS

El fichero `file_management.cpp` consta de las tres siguientes funciones:

- **Clear_file:** esta función se encarga de borrar el fichero. Recibe como parámetro el fichero a borrar y no devuelve nada. La secuencia de ejecución es la siguiente:
 1. Inicializa el SPIFFS.
 2. Abre el fichero en modo escritura.
 3. Cierra el archivo.

Al abrir el fichero en modo escritura y no escribir nada, el archivo queda borrado.

```
void clear_file(const char* measFile) {
    if (!SPIFFS.begin()) {
        Serial.println("[FILE_MGT] Error iniciando SPIFFS");
        return;
    }

    File file = SPIFFS.open(measFile, "w");
    if (!file) {
        Serial.println("[FILE_MGT] Error abriendo el fichero para borrar");
        return;
    }

    file.close();
    Serial.println("[FILE_MGT] El contenido del fichero ha sido borrado.");
}
```

Fragmento 1: Funcion `Clear_file`

- **Read_meas:** esta función realiza la operación de lectura del fichero. Recibe como parámetro el fichero a leer y no devuelve nada. La secuencia de ejecución es la siguiente:
 1. Inicia el SPIFFS.
 2. Abre el fichero en modo de lectura.
 3. Muestra por `Serial` el contenido del archivo hasta que detecta que acaba dicho fichero.
 4. Cierra el fichero.

```
void read_meas(const char* measFile){
    if (!SPIFFS.begin()) {
        Serial.println("[FILE_MGT] Error iniciando SPIFFS");
        return;
    }

    File measurementFile = SPIFFS.open(measFile, "r");
    if (!measurementFile) {
        Serial.println("[FILE_MGT] Error abriendo fichero");
        return;
    }
```

```

Serial.println("[FILE_MGT] Abriendo " + String(measFile) + ". Contenido: ");

while (measurementFile.available()) {
    Serial.write(measurementFile.read());
}

measurementFile.close();
}

```

Fragmento 2: Funcion Read_file

- **Write_meas:** esta función realiza la operación de escritura en el fichero. Recibe como parámetros el fichero a escribir, las medidas obtenidas por los sensores y una cadena de texto con la fecha y hora en la cual se han realizado dichas medidas y devuelve un booleano que indica si la operación se ha realizado de manera correcta. La secuencia de ejecución es la siguiente:

1. Inicia el SPIFFS.
2. Abre el fichero en modo **append** para escribir a continuación de la última medida y no sobrescribirla.
3. Se forma la cadena de texto que se desea escribir. Dicha cadena está formada por la fecha y hora de la medición y por los valores obtenidos.
4. Se escribe dicha cadena en el fichero.
5. Se cierra el fichero.
6. En caso de que la escritura haya sido satisfactoria, esta función devolverá true, en caso de que se haya encontrado un error en cualquiera de los pasos realizados, esta función devolverá false.

```

bool write_meas(const char* measFile, telemetry_t measures, String timestamp) {
    if (!SPIFFS.begin()) {
        return false;
    }

    File measurementFile = SPIFFS.open(measFile, "a");
    if (!measurementFile) {
        return false;
    }

    String measurement = timestamp +
        " VPanelSolar: " + String(measures.VSolar) + " V, " +
        " IPanelSolar: " + String(measures.ISolar) + " mA, " +
        " VBatBackup: " + String(measures.VBatbu) + " V, " +
        " IBatBackup: " + String(measures.IBatbu) + " mA, " +
        " VBat1: " + String(measures.VBat1) + " V, " +
        " IBat1: " + String(measures.IBat1) + " mA, " +
        " VBat2: " + String(measures.VBat2) + " V, " +
        " IBat2: " + String(measures.IBat2) + " mA";

    measurementFile.println(measurement);
    measurementFile.close();

    return true;
}

```

Fragmento 3: Funcion Write_file

3.2.2. Fichero Hardware.h

El fichero **Hardware.h** define los pines que utiliza el ESP.

Siendo estos pines:

- **GPIO13 y GPIO15:** Para la gestión de la conmutación de los relés.
- **GPIO4 y GPIO5:** Usados para la gestión de la comunicación I2C (Señales SDA y SCL respectivamente).

3.2.3. Módulo INA

El módulo INA es el encargado de realizar la configuración de los INA y solicitarles medidas. Este módulo está formado por dos archivos:

- **ina.cpp:** Contiene las funciones necesarias para la configuración de los sensores, el bajo consumo y la toma de medidas.
- **ina.hpp:** Contiene la definición de la función que solicita 1 única medida a los INA y las direcciones I2C de los INA.

Para la implementación de este módulo se ha requerido la utilización de la librería **INA266_WE.h**, que permite configurar y leer los datos mediante I2C con Arduino. [9]

El fichero **ina.cpp** contiene las siguientes funciones:

- **setupINA226Sensors():** Esta función inicializa los 4 INA con la configuración inicial que establece la librería **INA266_WE.h**.

```
void setupINA226Sensors() {
    Wire.begin();
    Solar.init();
    Batbu.init();
    Bat1.init();
    Bat2.init();
}
```

Fragmento 4: Código función **setupINA226Sensors**

- **reconfig_INAs():** Al despertar o inicializarlos INA, se encuentran con la configuración por defecto, por lo que es necesario reconfigurarlos acorde a las necesidades de nuestro circuito. Esta reconfiguración consta de lo siguiente:

- **setConversionTime:** Establece el tiempo que asignamos al INA para realizar la conversión de medidas físicas a su valor digital.
- **setMeasureMode:** Establece el modo de medida, **POWER_DOWN** (Sistema apagado), **TRIGGERED** (medidas a petición) o **CONTINUOUS** (medidas constantes).
- **setResistorRange:** Establece el valor de la resistencia de SHUNT que tiene el INA (Medido físicamente) y su rango de corriente con el que se va a trabajar.
- **setCorrectionFactor:** Establece el factor de corrección, el sensor nunca realiza una medición que corresponde con la realidad, por lo que es necesario indicar el factor de corrección que tiene que ser aplicado cuando se realizan estas medidas. Para el establecimiento de este valor se han realizado medidas experimentales midiendo el valor real y comparándolo con lo que indicado por el sensor.

```
void reconfig_INAs(){
    Solar.setConversionTime(CONV_TIME_1100);
    Batbu.setConversionTime(CONV_TIME_1100);
    Bat1.setConversionTime(CONV_TIME_1100);
    Bat2.setConversionTime(CONV_TIME_1100);

    Solar.setMeasureMode(CONTINUOUS);
    Batbu.setMeasureMode(CONTINUOUS);
    Bat1.setMeasureMode(CONTINUOUS);
    Bat2.setMeasureMode(CONTINUOUS);
```

```

Solar.setResistorRange(0.1, 10);
Batbu.setResistorRange(0.1, 10);
Bat1.setResistorRange(0.1, 10);
Bat2.setResistorRange(0.1, 10);

// Factores de corrección medidas experimentalmente
Solar.setCorrectionFactor(1.0469);
Batbu.setCorrectionFactor(1.0419);
Bat1.setCorrectionFactor(0.9650);
Bat2.setCorrectionFactor(0.9624);

Solar.waitUntilConversionCompleted();
Batbu.waitUntilConversionCompleted();
Bat1.waitUntilConversionCompleted();
Bat2.waitUntilConversionCompleted();
}

```

Fragmento 5: Código función reconfig_INAs

- `powerDownINA226()`: Pone en bajo consumo los INA, ya que tras conseguir las medidas no es necesario que se mantengan activos.

```

void powerDownINA226() {
    Solar.powerDown();
    Batbu.powerDown();
    Bat1.powerDown();
    Bat2.powerDown();
}

```

Fragmento 6: Código función powerDownINA226

- `powerUpINA226()`: Despierta de nuevo a los INA, al despertarlos los sensores se reinician con la configuración por defecto por lo que es necesario reconfigurarlos.

```

void powerUpINA226() {
    Solar.powerUp();
    Batbu.powerUp();
    Bat1.powerUp();
    Bat2.powerUp();
}

```

Fragmento 7: Código función powerUpINA226

- `measureINA226(telemetry_t *telemetry)`: Realiza la medida de todos los sensores. La secuencia de ejecución es la siguiente:

1. Verifica si se han inicializado previamente los INA. Si se han inicializado se despiertan.
2. Se reconfiguran.
3. Se toman las medidas de tensión y corriente de todos los INA y se guarda la información en el puntero con la estructura de datos.
4. Se ponen a bajo consumo los INA.

```

void measureINA226(telemetry_t *telemetry) {
    // Panel Solar
    if (first_config==false) {
        setupINA226Sensors();
        first_config=true;
    } else {
        powerUpINA226();
    }
}

```

```

    reconfig_INAs();

    Solar.readAndClearFlags();
    telemetry->VSolar = Solar.getBusVoltage_V() + (Solar.getShuntVoltage_mV() / 100);
    telemetry->ISolar = Solar.getCurrent_mA();

    // Bateria Backup
    Batbu.readAndClearFlags();
    telemetry->VBatbu = Batbu.getBusVoltage_V() + (Batbu.getShuntVoltage_mV() / 100);
    telemetry->IBatbu = -Batbu.getCurrent_mA();

    // Bateria 1
    Bat1.readAndClearFlags();
    telemetry->VBat1 = Bat1.getBusVoltage_V() + (Bat1.getShuntVoltage_mV() / 100);
    telemetry->IBat1 = -Bat1.getCurrent_mA(); // Invertir signo porque esta al revés

    // Bateria 2
    Bat2.readAndClearFlags();
    telemetry->VBat2 = Bat2.getBusVoltage_V() + (Bat2.getShuntVoltage_mV() / 100);
    telemetry->IBat2 = Bat2.getCurrent_mA();
    powerDownINA226();
}

```

Fragmento 8: Código función measureINA226

En el contenido de este mismo fichero, se instancian los 4 INA que tenemos indicando su dirección I2C:

```
#include "ina.hpp"

INA226_WE Solar(I2C_D_PANEL);
INA226_WE Batbu(I2C_D_BAT_BU);
INA226_WE Bat1(I2C_D_BAT_1);
INA226_WE Bat2(I2C_D_BAT_2);
```

Fragmento 9: Instancia de las direcciones de los INA.

3.2.4. Módulo Cliente MQTT

El módulo MQTT es el encargado de crear un cliente MQTT y establecer una conexión con el servidor MQTT para enviar los datos de la última medida que se haya realizado.

Este módulo está formado por dos archivos:

- `mqtt_client.cpp`: Contiene las funciones necesarias para la conexión.
- `mqtt_client.hpp`: Contiene la definición de las funciones, la importación de las librerías externas utilizadas y el número de reintentos permitidos al realizar una conexión con el servidor.

Para la implementación de este módulo se ha requerido la utilización de las librerías `PubSubClient.h`, que permite la conexión MQTT, `ESP8266WiFi.h` y `WiFiUdp.h` para configurar el cliente UDP. [10]

El fichero `mqtt_client.cpp` contiene las siguientes funciones:

- `reconnect(PubSubClient& client, char* const mqtt_server, int mqtt_port)`
Intentar una reconexión con el servidor MQTT en caso de que haya fallado. Se realiza un número limitado de reintentos antes de desistir.

```
void reconnect(PubSubClient& client, char* const mqtt_server, int mqtt_port) {
    for (int i = 0; i < 3 && !client.connected(); i++) {
```

```

        Serial.print("[MQTT] Iniciando conexión a " + String(mqtt_server) + ":" +
            String(mqtt_port));

        if (client.connect("Cliente", "Cliente", "Cliente")) {
            Serial.println("[MQTT] Conectado");
        } else {
            Serial.print("[MQTT] Conexión fallida. Estado = " + String(client.state()) +
                ". Reintentando en 0.5 segundos...");
            delay(500);
        }
    }
}

```

Fragmento 10: Función reconnect

■ `publishTelemetry(char* const mqtt_server, int mqtt_port, telemetry_t& telemetry)`.

Intenta publicar un mensaje con todos los datos medidos por el INA al servidor MQTT. La secuencia de ejecución es la siguiente:

1. Crea un cliente MQTT.
2. Crea una conexión con el servidor MQTT.
3. Verifica si el cliente se ha podido conectar con el servidor. Si no ha podido intenta realizar una reconexión en el caso de que haya habido un error.
4. Formatea el mensaje para que pueda ser procesado por el servidor.
5. Publica el mensaje en el servidor y este devuelve un booleano que indica si el envío ha sido correcto.
6. Se desconecta el cliente.
7. Devuelve el booleano.

```

bool publishTelemetry(char* const mqtt_server, int mqtt_port, telemetry_t& telemetry){
    WiFiClient wifiClient;
    WiFiUDP ntpUDP;
    PubSubClient client(wifiClient);
    client.setServer(mqtt_server, mqtt_port);
    bool statusPublish=false;
    char result[256];

    if (!client.connected()){
        reconnect(client, mqtt_server, mqtt_port);
    }

    sprintf(result, sizeof(result), "{VSolar: %.2f, ISolar: %.2f, VBatbu: %.2f, IBatbu: %.2f, "
        "VBat1: %.2f, IBat1: %.2f, VBat2: %.2f, IBat2: %.2f}",
        telemetry.VSolar, telemetry.ISolar, telemetry.VBatbu, telemetry.IBatbu,
        telemetry.VBat1, telemetry.IBat1, telemetry.VBat2, telemetry.IBat2);

    statusPublish = client.publish("vi/devices/me/telemetry" ,result);
    client.disconnect();
    return statusPublish;
}

```

Fragmento 11: Funcion publishTelemetry

3.2.5. Módulo Bajo Consumo

El módulo de bajo consumo es el encargado de reducir el consumo del sistema cuando no es necesario que funcione a plena potencia, es decir, entre las obtenciones de las diferentes medidas.

Las librerías incluidas con el **ESP8266** incluye un modo de bajo consumo ya definido llamado **deep-sleep**, el cual desactiva el **Wi-Fi**, el reloj del sistema y la CPU, es decir, solo mantiene el funcionamiento del RTC y reduce el consumo del sistema a apenas unas decenas de microamperios. Sin embargo, debido a que este modo de bajo consumo inhabilita también los **GPIO** es incompatible con nuestro sistema, ya que si se activara, el sistema no sería capaz de mantener los relés commutados al entrar a este modo. Debido a esto, este modo de bajo consumo no es apto para nuestro sistema. [11]

Item	Modem-sleep	Light-sleep	Deep-sleep
Wi-Fi	OFF	OFF	OFF
System clock	ON	OFF	OFF
RTC	ON	ON	ON
CPU	ON	Pending	OFF
Substrate current	15 mA	0.4 mA	~ 20 µA
Average current	DTIM = 1	16.2 mA	1.8 mA
	DTIM = 3	15.4 mA	0.9 mA
	DTIM = 10	15.2 mA	0.55 mA

Figura 35: Modos Bajo Consumo ESP8266

Por este requisito, hemos desarrollado nuestro propio modo de bajo consumo, el cual reduce el consumo del sistema en un 80 % aproximadamente. Este modo de bajo consumo es similar al llamado **modem-sleep**. Por nuestra parte, nuestro modo desactiva el **Wi-Fi**, baja la frecuencia del reloj del sistema y ejecuta un **delay**, para que el micro reduzca su consumo. Ya que en ningún momento se inhabilitan los **GPIO**, este modo de bajo consumo es perfecto para nuestro sistema.

Este módulo está dividido en dos ficheros, **sleep.hpp** el cual contiene la definición de la función de bajo consumo y la importación de la librería necesaria, y el fichero **sleep.cpp**, el cual contiene la implementación de dicha función, que recibe como parámetro el tiempo en milisegundos deseado de duración del modo de bajo consumo y no devuelve nada.

```
void sleep_low_power(int time_delay) {}
```

Fragmento 12: Función bajo consumo

La secuencia de ejecución se puede dividir en dos secuencias consecutivas, una que activa el modo de bajo consumo y otra que restaura el funcionamiento normal. La secuencia de activación es la siguiente:

1. Se desactiva el módulo de **Wi-Fi**, activando el modo **Wi-Fi_OFF**.
2. Se llama a la función **forceSleepBegin()**, la cual guarda el modo de **Wi-Fi** actual y fuerza el estado de modo **sleep** del **Wi-Fi**, reduciendo aún más el consumo.
3. Se reduce la frecuencia del reloj del sistema al mínimo valor posible, 80 MHz.
4. Se introduce un **delay**. Esto asegura que el microprocesador consumirá menos durante dicho tiempo, además se utiliza para definir la duración aproximada del modo de bajo consumo.

```
Serial.println("[SLEEP] Entrando en modo bajo consumo");
WiFi.mode(WIFI_OFF);
WiFi.forceSleepBegin();
system_update_cpu_freq(SYS_CPU_80MHZ);

delay(time_delay);
```

Fragmento 13: Activación modo bajo consumo

Una vez pasado el tiempo definido de bajo consumo, se entra en la secuencia de restauración, la cual consta de los siguientes pasos:

1. Aumentar la frecuencia del reloj al valor anterior, 160 MHz.
2. Forzar la activación del módulo del **Wi-Fi**.

3. Aplicar un `delay` de 1 ms para asegurar que el módulo Wi-Fi se haya despertado.
4. Configurar el módulo de Wi-Fi como una estación, es decir, configurar el ESP8266 como un dispositivo que se conecta a un punto de acceso.

```
Serial.println("[SLEEP] Saliendo del modo bajo consumo");
system_update_cpu_freq(SYS_CPU_160MHZ);
WiFi.forceSleepWake();
delay(1);
WiFi.mode(WIFI_STA);
```

Fragmento 14: Restauración modo bajo consumo

Mediante la implementación de nuestro propio modo de bajo consumo, nos aseguramos que correcto funcionamiento durante y después de dicho modo, reduciendo el consumo del sistema un 80 % aproximadamente, pasando de un consumo de 73 mA aproximadamente a pleno funcionamiento, a apenas 15 mA en el estado de bajo consumo.

3.2.6. Módulo Telemetría

El fichero `telemetry.hpp` define la estructura de datos que usaremos para la telemetría.

Dicha estructura almacenará los datos de las medidas obtenidas por los INA, es decir, los datos de tensión y corriente de las baterías (del usuario y *Backup*) y del panel solar.

```
typedef struct {
    float VSolar; /** Tension panel solar */
    float ISolar; /** Corriente panel solar */
    float VBatbu; /** Tension bateria backup */
    float IBatbu; /** Corriente bateria backup */
    float VBat1; /** Tension bateria 1 */
    float IBat1; /** Corriente bateria 1 */
    float VBat2; /** Tension bateria 2 */
    float IBat2; /** Corriente bateria 2 */
} telemetry_t;
```

Fragmento 15: Estructura de datos para la telemetría

3.3. Programa Principal

Este módulo está formado por el archivo, `MAIN_POWER.ino` que se encarga de conectar todos los módulos en un sistema unificado.

Inicialmente, en la cabecera del fichero se incluyen las librerías que vamos a utilizar:

- `telemetry.hpp`: Define la estructura de datos que será usada para almacenar las lecturas recibidas por los INA.
- `hardware.hpp`: Define los pines del ESP que son necesarias para el funcionamiento del sistema (I2C y relés).
- `ina.hpp`: Contiene el código relativo a la gestión, configuración y obtención de datos de los INA.
- `mqtt_client.hpp`: Contiene el código relativo a la conexión y envío de datos al servidor MQTT.
- `file_management.hpp`: Contiene el código relativo a la escritura en un fichero de los datos recibidos de los INA.
- `sleep.hpp`: Contiene el código que pone en bajo consumo al ESP.

```
#include "telemetry.hpp"
#include "hardware.hpp"
#include "ina.hpp"
```

```
#include "mqtt_client.hpp"
#include "file_management.hpp"
#include "sleep.hpp"
```

Fragmento 16: Cabecera con los módulos y ficheros utilizados

A continuación, se define una macro para imprimir mensajes de depuración por `Serial`, si la variable `DEBUG` está definida.

```
#define DEBUG

#ifndef DEBUG
#define PRINT_DEBUG(...) Serial.println(__VA_ARGS__)
#else
#define PRINT_DEBUG(...)
#endif
```

Fragmento 17: Definición de la macro de depuración

Esta macro se encarga de escribir por el puerto serie información de interés como:

- Mostrar la información recibida por los `INA` (así corroborar la correcta recepción en el `MQTT`).
- Indicar si se ha podido enviar correctamente los datos al servidor `MQTT`.
- Indicar la hora recibida del `NTP`.
- Indicar la correcta conexión con la red `Wi-Fi`.
- Indicar si se ha podido escribir correctamente en el fichero que almacena los datos de medidas.
- Indicar el estado del proceso de conexión con la `WiFi`.
- Indicar las modificaciones en los relés y el estado en el que se encuentran.

Después de este código, se encuentra la definición de distintos elementos utilizados por el `main`:

```
#define ssid          "POC"
#define password       "POWERIoT"

#define MQTT_SERVER    const_cast<char*>("192.168.111.69")
#define MQTT_PORT      4444

#define TIME_SLEEP     10000
#define WIFI_TRIES    35
#define BAT_THRESH     11.8
#define SOLAR_THRESH   20
#define MEASUREMENT_FILE "/measurements.txt"
```

Fragmento 18: Constantes de `MAIN_POWER`

- `SSID` y `password`: Son utilizados para indicar el `SSID` de la `Wi-Fi` a la que conectarse y su contraseña.
- `MQTT_SERVER` y `MQTT_PORT`: Indican la dirección y el puerto del servidor `MQTT` al que el `ESP` se intentará conectar para enviar datos.
- `TIME_SLEEP`: Indica el tiempo en ms que se quedará en bajo consumo/dormido el sistema.
- `WIFI_TRIES`: Indica el número de reintentos que damos al `ESP` para conectarse a la red `Wi-Fi`.
- `BAT_THRESH`: Umbral de la batería de backup con el que el sistema determina si activa la protección de sobrecarga.
- `SOLAR_THRESH`: Umbral del panel solar para el que el sistema determina si cambiamos a la carga mediante la batería de backup.

- **MEASUREMENT_FILE:** Indica la ruta donde se almacena el fichero de medidas.

El fichero contiene las siguientes funciones:

- **setup():** Inicializa los pines que usamos para la comunicación I2C y los relés, la velocidad del puerto serie y el valor inicial de la estructura de datos que contienen.

```
void setup() {
    pinMode(RELAY_SW_IN, OUTPUT);
    pinMode(RELAY_SW_OUT, OUTPUT);
    pinMode(SDA,          OUTPUT);
    pinMode(SCL,          OUTPUT);

    Serial.begin(9600);
}
```

Fragmento 19: Código de la función `setup`

- **loop():** Bucle principal de funcionamiento del sistema. La secuencia de ejecución es la siguiente:

1. Recoge las medidas de los INA.
 2. Revisa los valores obtenidos de tensión y actúa sobre los relés.
 3. Se intenta conectar a la red Wi-Fi. Si lo consigue se intenta mandar las medidas al servidor MQTT.
 4. Guarda las medidas de los sensores en un fichero.
 5. Duerme al ESP poniéndolo en bajo consumo.
 6. Sale del modo de bajo consumo.
-

```
measureINA226(&telemetry);

PRINT_DEBUG("[INA] VSolar [V]: " + String(telemetry.VSolar));
PRINT_DEBUG("[INA] ISolar [mA]: " + String(telemetry.ISolar));
PRINT_DEBUG("[INA] VBatbu [V]: " + String(telemetry.VBatbu));
PRINT_DEBUG("[INA] IBatbu [mA]: " + String(telemetry.IBatbu));
PRINT_DEBUG("[INA] VBat1 [V]: " + String(telemetry.VBat1));
PRINT_DEBUG("[INA] IBat1 [mA]: " + String(telemetry.IBat1));
PRINT_DEBUG("[INA] VBat2 [V]: " + String(telemetry.VBat2));
PRINT_DEBUG("[INA] IBat2 [mA]: " + String(telemetry.IBat2));

setRelays(&telemetry);

setup_wifi();

if (WiFi.status() == WL_CONNECTED){
    configTime(3600, 0, "time.nist.gov", "0.pool.ntp.org");

    if (!publishTelemetry(MQTT_SERVER, MQTT_PORT, telemetry)){
        PRINT_DEBUG("[POWER] ERROR publicando medidas");
    } else {
        PRINT_DEBUG("[POWER] Medidas publicadas con éxito");
    }

    getLocalTime(&currentTime, 1000);
    strftime(timeStr, 50, "[%d/%m/%y - %H:%M:%S]", &currentTime);
    PRINT_DEBUG("[WIFI] Sincronizado con NTP. Hora obtenida: " + String(timeStr));
} else {
    PRINT_DEBUG("[WIFI] No se ha podido conectar a la red WiFi");
    sprintf(timeStr, 50, "[00/00/00 - 00:00:00]");
}

if (write_meas(MEASUREMENT_FILE, telemetry, timeStr)){
```

```

    PRINT_DEBUG("[POWER] Fichero escrito con éxito");
} else {
    PRINT_DEBUG("[POWER] El fichero no ha podido ser escrito");
}

sleep_low_power(TIME_SLEEP);

```

Fragmento 20: Código de la función loop

La primera secuencia que hay en el código es `measureINA226(&telemetry)`; función tiene una ligera limitación, si los INA no están conectados el sistema se queda bloqueado. Esto ocurre ya que se queda a la espera de una respuesta de la dirección I₂C del INA al que está llamando, que en caso de no estar configurado, se quedará el sistema bloqueado hasta esa respuesta o un `reset`.

Tras recibir las medidas, si nos encontramos depurando el código, podremos observar por el puerto serie los datos que han sido almacenados de cada elemento del circuito.

Una vez leídas las medidas, el sistema verifica los valores de tensión del panel solar y de la batería y modifica los relés en función de estos. En caso de que no se supere un valor mínimo de tensión tanto del panel solar como de la batería de *backup*, el sistema activaría la protección de sobredescarga para evitar daños en el sistema. Entrar en este modo implicaría que las funciones del ESP dejarían de estar disponibles hasta que las tensiones superen los umbrales, alimentando una vez más al ESP y volviendo a su funcionamiento normal.

Tras esto, se intenta la conexión con la red Wi-Fi:

- No se consigue conectar a la Wi-Fi: Se establece la hora y fecha de las medidas realizadas a 0 y no se intenta enviar los datos al servidor MQTT.
- Si se consigue conectar a la Wi-Fi: Se configura un servidor NTP para asegurar la fecha y hora en las que realizan las medidas. Tras esto se intenta enviar los datos leídos al servidor MQTT.

Independientemente de si se ha podido conectar a la Wi-Fi o no, se intenta realizar una escritura con las lecturas realizadas en el fichero de medidas.

Finalmente, tras realizar todas las operaciones anteriores, se pone a dormir el ESP durante un tiempo TIME_SLEEP. Tras esto el sistema se vuelve a activar y a realizar las operaciones de nuevo.

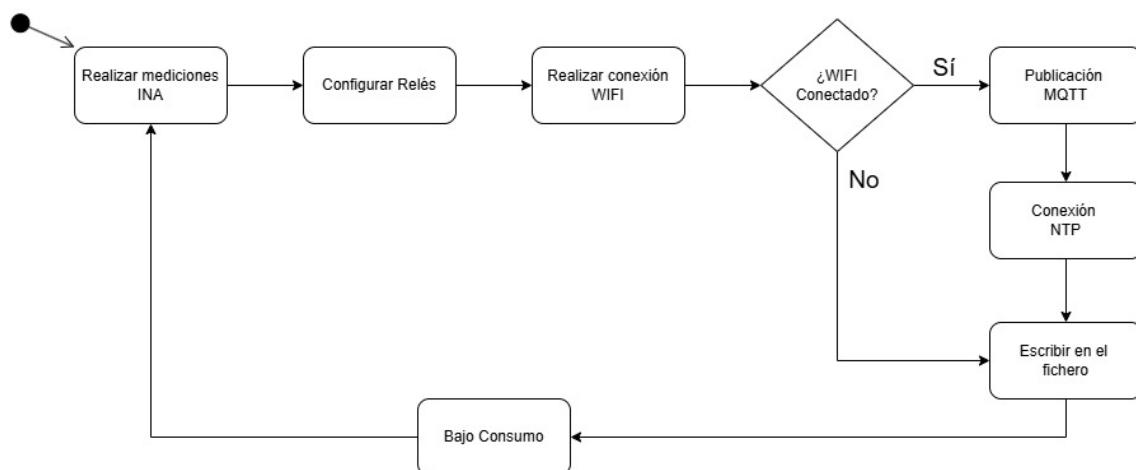


Figura 36: Diagrama de flujo Programa Principal

Para la implementación del programa principal, también se han desarrollado dos funciones adicionales que se encargan de intentar la conexión Wi-Fi y configurar los relés del sistema. A continuación, se explican dichas funciones:

- Función `setup_wifi()`: esta función es la encargada de la conexión Wi-Fi del sistema. Se intentará la conexión un número determinado de intentos, definidos en la variable `WIFI_TRIES`. Esta función no recibe ningún parámetro y tampoco devuelve nada.

```
static void setup_wifi();
```

Fragmento 21: Definición función setup_wifi

La secuencia de ejecución de esta función es la siguiente:

1. Muestra por serial un mensaje con la red Wi-Fi a la que se intenta la conexión.
 2. Intenta realizar la conexión a la red Wi-Fi con las credenciales almacenadas en las variables `ssid` y `password`. Estas variables almacenan el nombre de la red Wi-Fi y su correspondiente contraseña, respectivamente.
 3. Se intenta realizar la conexión un número finito de intentos indicados en la variable `WIFI_TRIES`. El tiempo de espera entre intentos de conexión es de 500 ms.
 4. En caso de que la conexión haya sido satisfactoria, el sistema mostrará por serial un mensaje al usuario indicando que la conexión Wi-Fi se ha realizado correctamente. En este mensaje se mostrará el nombre de la red a la que se ha realizado la conexión y la dirección IP de nuestro dispositivo ESP8266.
 5. Por otra parte, en caso de que el sistema no haya sido capaz de conectarse a la red Wi-Fi indicada, se mostrará un mensaje de error por serial indicando al usuario la conexión fallida.
-

```
static void setup_wifi() {
    delay(10);
    Serial.print("[WIFI] Intentando conectar a " ssid);
    WiFi.begin(ssid, password);
    for (int i = 0; i < WIFI_TRIES && WiFi.status() != WL_CONNECTED; i++) {
        delay(500);
        Serial.print(".");
        i++;
    }
    if (WiFi.status() == WL_CONNECTED) {
        PRINT_DEBUG("[WIFI] Conectado a la red WiFi " ssid " con IP: ");
        PRINT_DEBUG(WiFi.localIP());
    } else {
        PRINT_DEBUG("[WIFI] No se ha podido conectar a la red WiFi");
    }
}
```

Fragmento 22: Desarrollo función setup_wifi

- Función `setRelays`: esta función es la encargada de configurar los relés del sistema en función de los valores medidos por los sensores y de los valores almacenados en las variables `SOLAR_THRESH` y `BAT_THRESH`. Esta función recibe como parámetro un puntero a la variable `telemetry`, la cual almacena las medidas obtenidas por los diferentes sensores y no devuelve nada.
-

```
static void setRelays(telemetry_t *telemetry);
```

Fragmento 23: Definición función setRelays

Esta función se compone de tres casos para las diferentes posibilidades de funcionamiento del sistema. Estas opciones son las siguientes:

- $V_{Solar} > \text{SOLAR_THRESH}$: esta posibilidad se refleja cuando la tensión medida en el panel solar es mayor al valor del `threshold` solar, almacenado en dicha variable. Esto quiere decir que la energía aportada por el panel solar es suficiente para asegurar el correcto funcionamiento y cargar de forma adecuada tanto la batería de backup como las otras dos baterías de carga. En este caso, ambos relés están en la posición que hemos denominado `LOW`, conectando el panel solar como entrada de energía y habilitando la carga de la batería de *Backup*.
- $V_{Batbu} > \text{BAT_THRESH}$: en caso de que la tensión medida en el panel solar no sea suficiente para alimentar el circuito completo, se comprobará si la energía almacenada en la batería de

backup es suficiente. Para ello, se compara la tensión medida en dicha batería con su respectivo `threshold`, almacenado en la variable `BAT_THRESH`. En caso de que dicha tensión medida sea mayor, quiere decir que es suficiente para asegurar el correcto funcionamiento del sistema y la carga correcta de las otras dos baterías. Para este caso, los relés comutan a la posición denominada `HIGH`, la cual desconecta el panel solar del sistema y conecta la batería de backup al camino de alimentación del sistema.

- Por último, en caso de que la tensión solar no sea suficiente y la batería de *Backup* se descargue hasta el umbral, se debe para la descarga de la batería para protegerla, por lo que los relés volverán a la posición inicial (`LOW`) pero debido a que no hay alimentación suficiente, el sistema se apagará. El sistema permanecerá en este estado hasta que el panel solar vuelva a su funcionamiento nominal y pueda aportar la cantidad de energía suficiente para volver al comportamiento predeterminado del sistema, cargando tanto la batería de backup como las otras 2 baterías de carga.

```
static void setRelays(telemetry_t *telemetry){
    if (telemetry->VSolar > SOLAR_THRESH) {
        PRINT_DEBUG("[RELAY] Panel solar activo");

        digitalWrite(RELAY_SW_IN,LOW);
        digitalWrite(RELAY_SW_OUT,LOW);
    } else if (telemetry->VBatbu > BAT_THRESH){
        PRINT_DEBUG("[RELAY] Bateria backup activa");

        digitalWrite(RELAY_SW_IN,HIGH);
        digitalWrite(RELAY_SW_OUT,HIGH);
    } else {
        PRINT_DEBUG("[RELAY] Activando proteccion sobredescarga. Se detendra el sistema");

        digitalWrite(RELAY_SW_IN,LOW);
        digitalWrite(RELAY_SW_OUT,LOW);
    }
}
```

Fragmento 24: Desarrollo función `setRelays`

3.4. Programa Lectura Fichero

Para la lectura del fichero almacenado en la memoria FLASH del dispositivo, el cual contiene las medidas obtenidas por los sensores, se ha desarrollado un programa independiente el cual se encarga de realizar esta función. Además, permite la eliminación de dicho fichero. Para ejecutar dichas funciones, se enviará por serial al micro un carácter u otro mediante el teclado. Ambas operaciones se realizan mediante el sistema de archivos SPIFFS.

Para la realización de este programa, se ha utilizado las funciones `Clear_file` y `Read_meas` explicadas en el Subapartado 3.2.1, por lo que ha sido necesario importar el archivo `file_management.hpp`, el cual contiene las definiciones de dichas funciones.

Este programa se puede dividir en dos partes, la función `setup()`, la cual se ejecuta tan solo una vez al arrancar el programa, y la función `loop()`, la cual consiste en un bucle infinito con el principal funcionamiento del programa. A continuación, se va a explicar el funcionamiento de estas dos funciones:

- `setup()`: Esta función tan solo inicializa el serial a una velocidad de 9600 *bps*, y muestra el siguiente mensaje por serial al usuario:

```
[LOG_MGR] Envie 'b' o 'd' para borrar el archivo y 'r' o 'l' para leerlo
```

Figura 37: Mensaje inicial programa de lectura

- **loop()**: Contiene el principal funcionamiento del programa y su secuencia de ejecución es la siguiente:
 1. Verificar si el serial funciona correctamente. En caso de que el serial no esté en pleno funcionamiento, el programa no hace nada.
 2. Lee el carácter enviado mediante el serial y lo guarda en una variable.
 3. En caso de que dicho carácter sea una **b** o una **d**, se procede a la secuencia de eliminación del fichero. Dicha secuencia es la siguiente:
 - a) Se muestra por serial un mensaje indicando al usuario la acción realizada.
 - b) Se llama a la función **clear_file**, incluida en la librería **file_management.hpp**, pasándole como parámetro el archivo a eliminar.
 - c) Una vez eliminado el fichero, se realiza una comprobación de la existencia de dicho archivo, mostrando por serial el estado del archivo.
 - d) Se cierra el archivo.
 4. Por otra parte, en caso de que el carácter leído haya sido una '**r**' o una '**l**', se procede a la lectura del fichero, llamando a la función **read_meas**, en la librería **file_management.hpp**, pasándole como parámetro el archivo a leer.
 5. En caso de que el carácter no haya sido ninguno de los anteriores mencionados, el sistema mostrará por serial el mismo mensaje que al principio de la ejecución de este programa, indicando al usuario que caracteres son aceptados.

```
#include "file_management.hpp"

const char* testFile = "/measurements.txt";

void setup() {
  Serial.begin(9600);
  Serial.println("[LOG_MGR] Envie 'b' o 'd' para borrar el archivo y 'r' o 'l' para
  leerlo ");
}

void loop() {
  // Verificar si hay datos disponibles en el puerto serial
  if (Serial.available() > 0) {
    char command = Serial.read(); // Leer el carácter ingresado
    if (command == 'b' || command == 'd') {

      Serial.println("[LOG_MGR]Borrando contenido del archivo...");
      clear_file(testFile);

      // Confirmar estado del archivo despues de borrarlo
      File file = SPIFFS.open(testFile, "r");
      if (file && file.available()) {
        Serial.println("[LOG_MGR] El archivo aun tiene contenido.");
      } else {
        Serial.println("[LOG_MGR] El archivo esta vacio.");
      }
      file.close();
    } else if (command == 'r' || command == 'l') {
      read_meas(testFile);
    } else if (command != '\n') {
      Serial.println("[LOG_MGR] Envie 'b' o 'd' para borrar el archivo y 'r' o 'l'
      para leerlo ");
    }
  }
}
```

Fragmento 25: Programa lectura o eliminación de fichero

3.5. Librerías Externas

3.5.1. Librería INA

Una de las librerías utilizadas para realizar el proyecto ha sido la librería **INA226_WE**. Esta contiene toda la información necesaria para la comunicación I₂C, configuración y obtención de datos de los sensores **INA226**, además de otras herramientas útiles como el establecimiento del modo de bajo consumo para los sensores.

En nuestro proyecto hemos utilizado esta herramienta para obtener los datos de tensión y corriente de los 4 **INA**. Aunque adicionalmente, se han utilizado las herramientas de configuración para ajustar de manera correcta los sensores y que estas medidas obtenidas sean fieles a la realidad, además de utilizar su modo de bajo consumo cuando no era necesario obtener medidas.

La versión utilizada es la **1.2.9** y pueden obtenerse los ficheros fuente en su repositorio en [GitHub](#). [9]

Adicionalmente, se cuenta con una página con documentación completa de esta librería y su funcionamiento. [12]

3.5.2. Librería PubSubClient

La librería utilizada para implementar el cliente MQTT ha sido la librería **PubSubClient**. Esta contiene toda la información necesaria para la creación de un cliente MQTT, la conexión con un servidor MQTT y la publicación y recepción de mensajes de servidores MQTT.

En nuestro proyecto hemos utilizado esta herramienta únicamente para la conexión con el servidor MQTT y el envío de los datos de las medidas de los **INA**.

La versión utilizada es la **2.8** y pueden obtenerse los ficheros fuente en su repositorio en [GitHub](#). [10]

Adicionalmente, se cuenta con una página con documentación completa de esta librería y su funcionamiento. [13]

3.6. Documentación del código

Debido a la alta modularidad del código y como se ha explicado previamente, ha sido dividido en módulos. Para simplificar la integración y legibilidad de dichos módulos, se han comentado extensivamente y se utiliza **Doxygen** para generar la página de la documentación en formato HTML.

Para ello, se deben utilizar comentarios con un formato muy marcado como el que se ve en el siguiente fragmento:

```
 /**
 * @brief Escribe la medida en el fichero indicado.
 * Formato: [timestamp] VPanelSolar: X.XX V, IPanelSolar: X.XX mA, VBatBackup: X.XX V,
 *           IBatBackup: X.XX mA, VBat1: X.XX V, IBat1: X.XX mA, VBat2: X.XX V, IBat2: X.XX mA
 *
 * @param measFile Nombre del fichero.
 * @param measures Medidas a escribir.
 * @param timestamp Marca de tiempo.
 * @return true si la escritura fue exitosa, false en caso contrario.
 */
extern bool write_meas(const char* measFile , telemetry_t measures, String timestamp);
```

Fragmento 26: Ejemplo de comentario Doxygen

Gracias a este formato estándar, se puede extraer toda la documentación y generar una web (<https://david-andrino.github.io/iot-power>) como la que se ve en la siguiente figura.

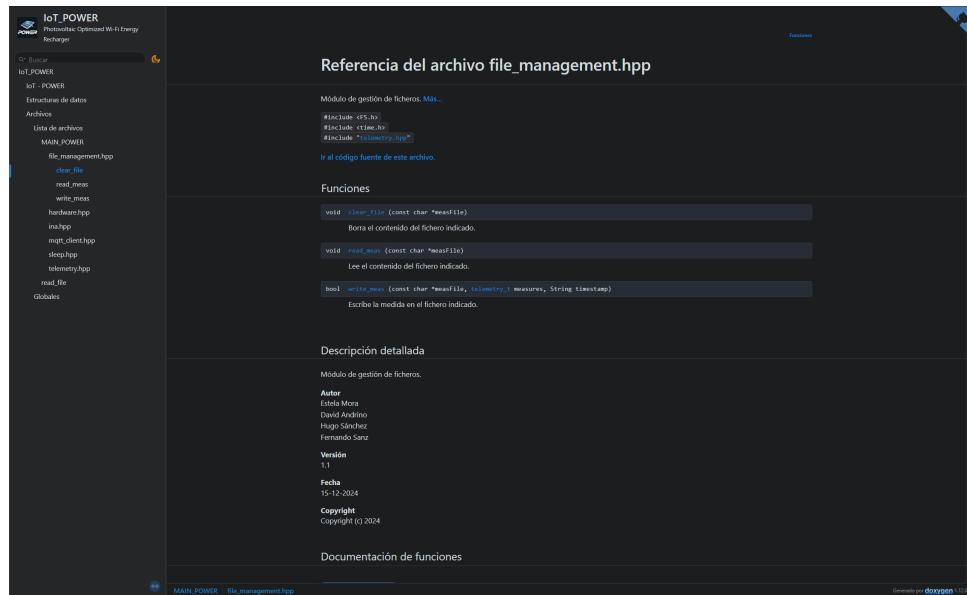


Figura 38: Captura de la web de documentación

Para automatizar la generación, se ha utilizado una **GitHub Action**, la cual genera automáticamente la documentación y la publica gracias a **GitHub Pages**. Esta acción y el tema de la documentación se han obtenido del repositorio **doxygen-awesome-css**. [14]

Estas herramientas de CI/CD permiten agilizar altamente el desarrollo de dicha documentación, ya que esta se regenera y publica automáticamente después de cada **Commit** de la rama principal.

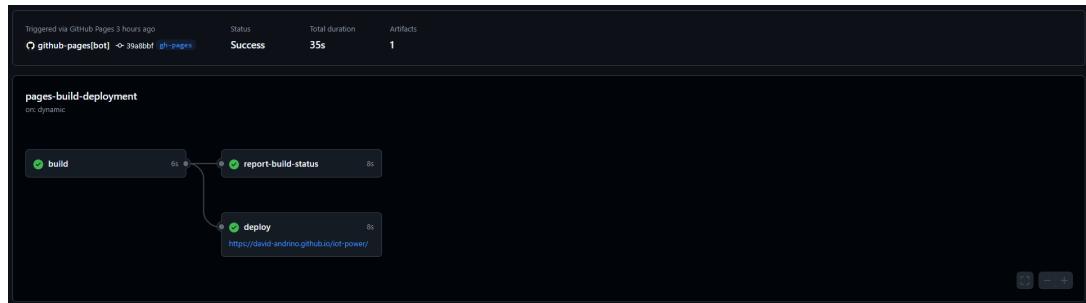


Figura 39: GitHub Action para la generación de documentación

4. Consideraciones teóricas

4.1. Panel solar

Para la elección del panel solar, es necesario analizar las condiciones mínimas de funcionamiento del sistema. Mediante las pruebas realizadas en el laboratorio, se ha determinado que el sistema necesita un voltaje mínimo de 20V en la entrada del panel solar. Por otra parte, se ha medido que el sistema, a pleno funcionamiento cargando las 3 baterías, consume unos 700mA, por lo que, añadiendo un 30 % de posibles pérdidas, se ha determinado que el panel solar debe aportar unos 910mA como mínimo para asegurar el correcto funcionamiento del sistema.

Debido a estas restricciones, se ha decidido utilizar el panel solar SOLARIMPUT PF200T del fabricante SEMPLER SOLUTIONS. Este es un panel solar monocristalino, lo cual nos asegura una mayor eficiencia y mayor rendimiento, lo que se refleja en una mayor cantidad de energía con la misma cantidad de luz. Por otra parte, los monocristalinos tienden a ser más duraderos, ofreciendo una mayor resistencia a la sombra y al viento. [15]

En cuanto al panel seleccionado, vamos a discutir los diferentes aspectos analizados para asegurar el correcto funcionamiento del sistema:

- **Compatibilidad con el sistema de carga:** Este panel ofrece un voltaje de máxima potencia V_{MP} de 20,88V y una corriente de máxima potencia I_{MP} de aproximadamente 9,58A. Estas especificaciones coinciden con los requisitos de nuestro sistema de carga, que incluye tres baterías de 12V y 7Ah conectadas en paralelo.

Su voltaje en circuito abierto V_{OC} de 24,48V asegura una operación eficiente con el controlador de carga, incluso bajo condiciones de luz reducida.

- **Eficiencia y tamaño:** Al ser un panel monocristalino, como se ha explicado anteriormente, el SOLARIMPUT PF200T ofrece una mayor eficiencia en comparación con paneles policristalinos de similar potencia. Esto permite un aprovechamiento máximo de la energía solar en espacios limitados.

Además, su diseño compacto y ligero facilita la instalación en espacios reducidos, una característica clave para proyectos donde el espacio disponible es limitado.

- **Durabilidad y fiabilidad:** Este panel está fabricado con materiales de alta calidad que le otorgan resistencia a condiciones climáticas adversas. Su estructura incluye un marco de aluminio anodizado y una cubierta de vidrio templado que protegen las celdas solares de impactos y agentes externos, garantizando una larga vida útil.

- **Prestaciones del panel:**

Células Solares	Monocristalino N-Type TOPcon
Potencia Máxima	200W
Tensión Máxima Potencia	20.88V
Tensión De Circuito Abierto	24.48V
Eficiencia Celular	22%
Medidas Desplegadas	1325x760x30mm
Peso	7.3kg
Amperios Máximos De Salida IMP	9.58A
Corriente En Cortocircuito ISC	10.15A
Estructura Del Módulo	Aluminio / Cristal / EVA / Backsheet
Caja De Conexiones	IP67
Conectores	MC4 Compatibles

Figura 40: Prestaciones SOLARIMPUT PF200T

Todas estas características se han obtenido de la hoja de características aportada por el fabricante. [16]

Por otra parte, para garantizar la eficiencia y seguridad del sistema, hemos decidido incorporar el regulador de carga solar LandStar LS1524. Este componente aporta un papel importante en la regulación del flujo de energía entre el panel solar y las baterías, previniendo sobredescargas, sobrecargas y garantizando más tiempo de vida útil al sistema.

De forma análoga al desarrollo realizado con el panel solar, a continuación se van a discutir las diferentes características de este regulador de carga solar.

- **Compatibilidad con el sistema de carga:** El regulador de carga LandStar LS1524 es compatible con sistemas de 12V y 24V lo que lo hace compatible con el panel solar seleccionado, el cual tiene una V_{MP} de 20,88V. Además, tiene la capacidad de manejar corrientes de hasta 15A, más que suficiente para los 9,58A de I_{MP} que aporta nuestro panel solar.

- **Características técnicas:**

- **Regulación:** Este controlador utiliza regulación PWM o Modulación por Ancho de Pulso. Este método asegura una carga eficiente y una menor pérdida de energía durante la transferencia, actuando como un interruptor entre el panel y el circuito, forzando al módulo fotovoltaico a trabajar a la tensión deseada sin ningún tipo de instalación extra.
- **Protección:** Este modelo incluye protección contra cortocircuitos, sobredescarga y sobrecarga, asegurando la seguridad del panel solar y del circuito de carga.
- **Indicadores:** Este regulador incorpora diferentes LED, aportando información visual sobre el estado de carga y posibles fallos, facilitando el monitoreo del sistema.
- **Eficiencia y robustez:** El controlador está diseñado para funcionar de manera eficiente en condiciones ambientales adversas. Además, su carcasa resistente y su fiable diseño electrónico, nos aseguran un correcto funcionamiento en aplicaciones en exteriores.

- **Prestaciones del regulador:**

Electrical parameters	LS1024	LS1524	LS2024
Nominal System Voltage	12 / 24VDC auto work		
Rated Battery Current	10A	15A	20A
Max. Battery Voltage	32V		
Charge Circuit Voltage Drop	$\leq 0.26V$		
Discharge Circuit Voltage Drop	$\leq 0.15V$		
Self-consumption	$\leq 6mA$		

Figura 41: Prestaciones LandStar LS1524

Todas estas características se han obtenido de la hoja de características aportada por el fabricante. [17]

En conclusión, la combinación del panel solar SOLARIMPUT PF200T y el regulador de carga solar LandStar LS1524, constituyen una solución eficiente y fiable para los requisitos de nuestro sistema de carga de baterías. Por una parte, el panel solar ofrece las características técnicas necesarias para aportar la energía requerida por el sistema, mientras que el regulador de carga solar ofrece una correcta gestión de dicha energía y aporta una serie de protecciones que hacen al sistema más robusto y más seguro frente a posibles fallos. Ambos componentes garantizan una gran eficiencia, estabilidad y durabilidad al sistema, cumpliendo con los requisitos mínimos estimados y ofreciendo un gran equilibrio entre prestaciones y costes. Obtenido un flujo de corriente como el siguiente:



Figura 42: Flujo de Corriente Panel Solar

4.2. Caja

En este apartado se realizará un desarrollo teórico de la caja que se utilizaría para desarrollar la solución.

4.2.1. Requisitos técnicos

Para asegurar una protección completa de todo el sistema, se ha pensado en desarrollar una caja que mantenga protegido a todo el circuito y al sistema de carga de cualquier elemento externo que pueda perturbar su funcionamiento.

Esta protección debe asegurar las siguientes condiciones:

1. **Protección contra agua y partículas:** Para evitar la entrada de agua dentro de la caja pudiendo generar cortocircuitos o dañar gravemente el funcionamiento del sistema. Estos daños también pueden ser ocasionados por la entrada de polvo u otras pequeñas partículas dentro de la caja.
2. **Acceso restringido:** Permitir el acceso a la carga de baterías solo se permitirá para el cliente, mientras que el acceso al circuito de carga y a la batería de backup.
3. **Aislamiento eléctrico:** Asegurar un aislamiento eléctrico de factores externos para evitar interferencias con el sistema.
4. **Resistencia a golpes:** El sistema al encontrarse en un entorno al aire libre es necesario proveerlo de cierta capacidad de protección en caso de golpes o impactos de factores externos.
5. **Resistencia rayos ultravioleta:** El sistema se carga principalmente mediante energía solar, esto implica que el sistema de carga se ve expuesto constantemente al sol y por ende a rayos ultravioleta, lo que puede dañar materiales como los polímeros.
6. **Temperaturas de trabajo:** La caja deberá ser capaz de aguantar altas temperaturas exteriores durante largos periodos de tiempo debido al entorno al que está expuesto.

4.2.2. Diseño 3D

Para tener un diseño preliminar de cómo sería la caja real se ha diseñado mediante el software de FreeCAD el siguiente modelo en 3D:

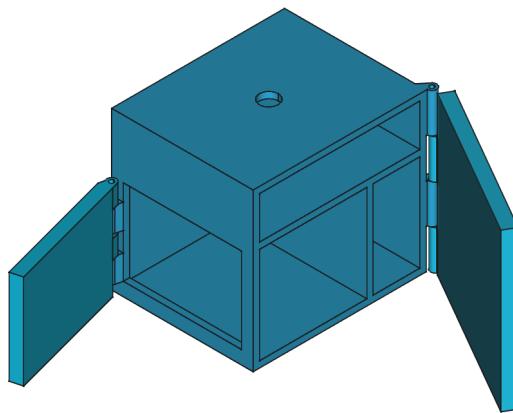


Figura 43: Vista Isométrica Modelo 3D caja

Como se puede observar, la caja cuenta con 2 accesos:

- **Puerta de usuario:** Esta puerta es usada por el cliente para únicamente conectar/desconectar las baterías que se desean cargar.

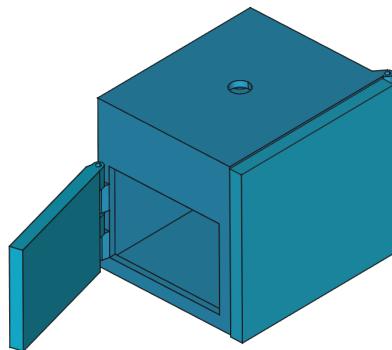


Figura 44: Vista Puerta Usuario Modelo 3D caja

- **Puerta de técnico:** Esta puerta es usada por el personal de administración para acceder tanto al circuito de carga como las baterías de backup y que estén cargándose.

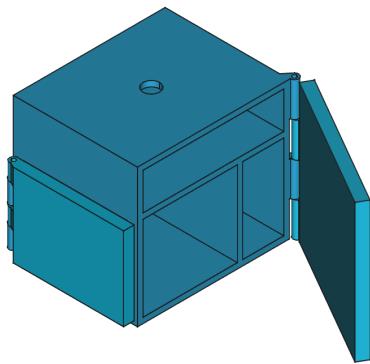


Figura 45: Vista Puerta Técnico Modelo 3D caja

Adicionalmente, el interior de la caja tiene 3 incisiones. Estas serán usadas únicamente para dar acceso a los terminales de carga para las baterías del usuario (los 2 accesos de la izquierda) y la de backup (acceso de la derecha).

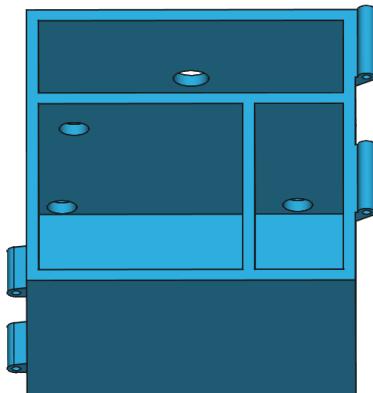


Figura 46: Vista Incisiones Modelo 3D caja

Además, en la parte superior de la caja contamos con un acceso para la entrada del cableado del panel solar para la caja.

Nota: Para facilitar el entendimiento del diseño en el Apéndice B se encuentran en detalle todas las vistas del diseño e imágenes adicionales del mismo.

4.2.3. Características técnicas

Para cumplir el diseño teórico se proponen las siguientes características técnicas:

- **Protección contra agua y partículas:** La caja cumplirá la norma internacional CEI 60529 para permitir esta protección. En concreto una protección IP67 que garantiza una protección completa contra polvo y pequeñas partículas y además, la inmersión completa de la caja a sin filtración alguna. [18]
- **Acceso restringido:** Esto se logra con el diseño 3D que hemos propuesto para la caja, la única implementación adicional que es necesaria añadir son cerraduras en cada una de las puertas.
- **Aislamiento eléctrico, resistencia rayos ultravioleta y temperatura:** Para esto se plantea que el material de la caja sea de policarbonato. Gracias a esto se consigue evitar el efecto de jaula de Faraday que pudiera causar el uso de una caja de metal. Además, el policarbonato proporciona una protección total de los rayos ultravioleta y una temperatura de operación desde -40°C hasta 115°C siendo un rango más que de sobra para nuestro circuito. [19] Además, protege de posibles cortos en caso de que alguno de los elementos de la caja hiciera contacto con ella.

Como elemento adicional, todo el cableado expuesto tanto en el interior de la caja (cableado usado por el usuario) como exteriores (cableado del panel solar) pasarán por prensaestopas y el cableado exterior constará de una funda de protección para el paso del panel a la caja.



Figura 47: Funda cableado exterior



Figura 48: Prensaestopa

- **Resistencia a golpes:** En caso de impactos la caja debe proporcionar cierta protección, para esto debe de asegurar un índice de protección IK entre 08 y 09 que ayudaría a mitigar los golpes de efectos adversos como el granizo o golpes de pequeñas rocas. [18]

5. Presupuesto

En la Tabla 4 se pueden ver los gastos del proyecto.

Componente	Precio unitario	Cantidad	Precio total
Batería	12,00 €	2	24,00 €
CN3768	5,89 €	3	17,67 €
LDO AMS1117	2,05 €	1	2,05 €
INA226	1,75 €	4	7,00 €
PS58F1	4,29 €	1	4,29 €
LM2596	0,61 €	1	0,61 €
XL6009	0,79 €	1	0,79 €
Componentes para el montaje	20,00 €	N/A	20,00 €
PCB relé	1,00 €	5	5,00 €
Componentes relé	5,00 €	N/A	5,00 €
Total			86,41 €

Tabla 4: Presupuesto del proyecto

6. Horas dedicadas al proyecto

En la Tabla 5 se pueden ver las principales tareas en las que se ha dividido el proyecto y las horas que han tomado cada una.

Integrante	Tareas	Horas
David	Diseño de sistema electrónico	7
	Selección de componentes electrónicos	2
	Caracterización de componentes	5
	Diseño de PCB de relés	5
	Montaje de prototipo electrónico	50
	Depuración de circuito electrónico	40
	Memoria	20
Estela	Diseño de sistema electrónico	7
	Compra de componentes	2
	Caracterización de componentes	5
	Diseño de PCB de relés	5
	Montaje de prototipo electrónico	50
	Depuración de circuito electrónico	40
	Memoria	20
Hugo	Búsqueda librerías compatibles con INA	3
	Código INA	3
	Código Wi-Fi/MQTT en Arduino	7
	Pruebas de los módulos individuales de programación	4
	Regulación y ajuste de los INA	2
	Desarrollo de la caja en 3D	10
	Ánálisis de características y requisitos de protección de la caja	15
	Memoria	16
	Total	350

Tabla 5: Distribución de horas dedicadas al proyecto

Como se puede ver en la tabla, se han dedicado muchas más horas de las estimadas inicialmente (200). Esto es debido principalmente a la elevada complejidad del circuito electrónico y la alta cantidad de componentes que interconectar, caracterizar y probar.

7. Bibliografía

- [1] Espressif Systems, «ESPRESSIF SMART CONNECTIVITY PLATFORM: ESP8266,» inf. téc. visitado 23 de dic. de 2024. dirección: https://www.elecrow.com/download/ESP8266_Specifications_English.pdf.
- [2] CONSONANCE, «CN3768,» inf. téc. visitado 17 de dic. de 2024. dirección: <http://www.alldata-sheet.es/datasheet-pdf/view/1133237/CONSONANCE/CN3768.html>.
- [3] Texas Instruments, «INA226 36V, 16-Bit, Ultra-Precise I2C Output Current, Voltage, and Power Monitor With Alert,» inf. téc. visitado 23 de dic. de 2024. dirección: https://www.ti.com/lit/ds/symlink/ina226.pdf?ts=1734980356982&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252Fes-mx%252FINA226.
- [4] XLSEMI, «400KHz 60 V 4 A Switching Current Boost / Buck-Boost / Inverting DC/DC Converter,» inf. téc. visitado 23 de dic. de 2024. dirección: <https://www.pollin.de/productdownloads/D351434D.PDF>.
- [5] Texas Instruments, «LM2596 SIMPLE SWITCHER® Power Converter 150-kHz 3-A Step-Down Voltage Regulator,» inf. téc. visitado 23 de dic. de 2024. dirección: https://www.ti.com/lit/ds/symlink/lm2596.pdf?ts=1734981821454&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252Fes-mx%252FLM2596.
- [6] Advanced Monolithic Systems, «AMS1117,» inf. téc. dirección: <http://www.advanced-monolithic.com/pdf/ds1117.pdf>.
- [7] thingsboard, *Installing ThingsBoard CE on Ubuntu Server*. visitado 14 de dic. de 2024. dirección: <https://thingsboard.io/docs/user-guide/install/ubuntu/>.
- [8] thingsboard, *MQTT Device API Reference*. visitado 14 de dic. de 2024. dirección: <https://thingsboard.io/docs/reference/mqtt-api/>.
- [9] W. (Ewald, *Wollewald/INA226-WE*, dic. de 2024. visitado 26 de dic. de 2024. dirección: https://github.com/wollewald/INA226_WE.
- [10] N. O'Leary, *Knolleary/Pubsubclient*, dic. de 2024. visitado 26 de dic. de 2024. dirección: <https://github.com/knolleary/pubsubclient>.
- [11] ESP8266, «ESP8266 Low Power Solutions,» inf. téc. visitado 23 de dic. de 2024. dirección: https://www.espressif.com/sites/default/files/9b-esp8266-low_power_solutions_en_0.pdf.
- [12] W. Ewald, *INA226 Current and Power Sensor • Wolles Elektronikkiste*, ene. de 2021. visitado 28 de dic. de 2024. dirección: <https://wolles-elektronikkiste.de/en/ina226-current-and-power-sensor>.
- [13] Nick O'Leary, *Arduino Client for MQTT*. visitado 28 de dic. de 2024. dirección: <https://pubsubclient.knolleary.net/api>.
- [14] jotheapro, *Jotheapro/Doxygen-Awesome-Css*, dic. de 2024. visitado 27 de dic. de 2024.
- [15] AutoSolar, *Diferencias entre silicio monocristalino y policristalino*. visitado 23 de dic. de 2024. dirección: <https://autosolar.es/aspectos-tecnicos/diferencias-entre-silicio-monocristalino-y-multicristalino-o-policristalino>.
- [16] Sempere Solutions, *Panel Solar Monocristalino 200W — Entrega 24h*. visitado 23 de dic. de 2024. dirección: <https://www.semperesolutions.com/solarimput-pf200-panel-solar-fijo-monocristalino-24v-200w>.
- [17] EPSolar, «LandStar series Solar Controller,» inf. téc. visitado 23 de dic. de 2024. dirección: <https://luminasol.com.mx/wp-content/uploads/2016/05/LS1024-LS2024-Specifications.pdf>.
- [18] IEC, *International Standard IEC 60529*. visitado 26 de dic. de 2024. dirección: <https://www.wewontech.com/IEC60529-%20IP-Standard.pdf>.
- [19] T. Horesh, *Polycarbonate and Protection from UV Radiation. What does this really mean?* Ago. de 2021. visitado 26 de dic. de 2024. dirección: <https://www.palram.com/blog/construction-architecture/polycarbonate-and-protection-from-uv-radiation/>.

Anexo A Código de la aplicación

```


/** 
 * @file file_management.hpp
 * @author Estela Mora
 * @author David Andriño
 * @author Hugo Sanchez
 * @author Fernando Sanz
 * @brief Modulo de gestion de ficheros
 * @version 1.1
 * @date 15-12-2024
 *
 * @copyright Copyright (c) 2024
 *
 */
#ifndef FILE_MANAGEMENT_H
#define FILE_MANAGEMENT_H

#include <FS.h>
#include <time.h>
#include "telemetry.hpp"

/** 
 * @brief Borra el contenido del fichero indicado.
 *
 * @param measFile Nombre del fichero a borrar.
 */
extern void clear_file(const char* measFile);

/** 
 * @brief Lee el contenido del fichero indicado. Imprime en consola el contenido.
 *
 * @param measFile Nombre del fichero a leer.
 */
extern void read_meas(const char* measFile);

/** 
 * @brief Escribe la medida en el fichero indicado.
 * Formato: [timestamp] VPanelSolar: X.XX V, IPanelSolar: X.XX mA, VBatBackup: X.XX V,
 *          IBatBackup: X.XX mA, VBat1: X.XX V, IBat1: X.XX mA, VBat2: X.XX V, IBat2: X.XX mA
 *
 * @param measFile Nombre del fichero.
 * @param measures Medidas a escribir.
 * @param timestamp Marca de tiempo.
 * @return true si la escritura fue exitosa, false en caso contrario.
 */
extern bool write_meas(const char* measFile , telemetry_t measures, String timestamp);

#endif


```

Fragmento 27: Fichero file_management.hpp

```


#include "file_management.hpp"

void clear_file(const char* measFile) {
    if (!SPIFFS.begin()) {
        Serial.println("[FILE_MGT] Error iniciando SPIFFS");
        return;
    }

    File file = SPIFFS.open(measFile, "w");
    if (!file) {


```

```

        Serial.println("[FILE_MGT] Error abriendo el fichero para borrar");
        return;
    }

    file.close();
    Serial.println("[FILE_MGT] El contenido del fichero ha sido borrado.");
}

void read_meas(const char* measFile){
    if (!SPIFFS.begin()) {
        Serial.println("[FILE_MGT] Error iniciando SPIFFS");
        return;
    }

    File measurementFile = SPIFFS.open(measFile, "r");
    if (!measurementFile) {
        Serial.println("[FILE_MGT] Error abriendo fichero");
        return;
    }

    Serial.println("[FILE_MGT] Abriendo " + String(measFile) + ". Contenido: ");

    while (measurementFile.available()) {
        Serial.write(measurementFile.read());
    }

    measurementFile.close();
}

bool write_meas(const char* measFile, telemetry_t measures, String timestamp) {
    if (!SPIFFS.begin()) {
        return false;
    }

    File measurementFile = SPIFFS.open(measFile, "a");
    if (!measurementFile) {
        return false;
    }

    String measurement = timestamp +
        " VPanelSolar: " + String(measures.VSolar) + " V, " +
        " IPanelSolar: " + String(measures.ISolar) + " mA, " +
        " VBatBackup: " + String(measures.VBatbu) + " V, " +
        " IBatBackup: " + String(measures.IBatbu) + " mA, " +
        " VBat1: " + String(measures.VBat1) + " V, " +
        " IBat1: " + String(measures.IBat1) + " mA, " +
        " VBat2: " + String(measures.VBat2) + " V, " +
        " IBat2: " + String(measures.IBat2) + " mA";

    measurementFile.println(measurement);
    measurementFile.close();

    return true;
}

```

Fragmento 28: Fichero file_management.cpp

```

/**
 * @file hardware.hpp
 * @author Estela Mora
 * @author David Andriño
 * @author Hugo Sanchez
 * @author Fernando Sanz

```

```

* @brief Definicion de pines hardware
* @version 1.1
* @date 15-12-2024
*/
#ifndef HARDWARE_H
#define HARDWARE_H

#define RELAY_SW_IN 13
#define RELAY_SW_OUT 15
#define SDA 4
#define SCL 5

#endif

```

Fragmento 29: Ficherohardware.hpp

```

///! Clase ina. Engargada de configurar y solicitar medidas al los INAs del circuito.

#include "ina.hpp"

// INA226 instances
INA226_WE Solar(I2C_D_PANEL);
INA226_WE Batbu(I2C_D_BAT_BU);
INA226_WE Bat1(I2C_D_BAT_1);
INA226_WE Bat2(I2C_D_BAT_2);

///! Booleano.
/*! Indica si se ha realizado la configuracion inicial de los sensores*/
bool first_config=false;

void setupINA226Sensors();
void reconfig_INAs();
void powerDownINA226();
void powerUpINA226();

/********************* */
 * Configuracion inicial de los sensores.
 *
 * Tras un reset se inician los sensores para ser modificados, estableciendose los parametros:
 * -Tiempo de conversion: Entre 140 us y 8.244 ms.
 * -Modo de medida: POWER_DOWN (Sistema apagado), TRIGGERED (medidas a peticion) o CONTINUOUS
 * (medidas constantes).
 * -Valor de la resistencia se SHUNT.
 * -Rango de corrientes.
 * -Factor de correccion.
/********************* /
void setupINA226Sensors() {
    Wire.begin();
    Solar.init();
    Batbu.init();
    Bat1.init();
    Bat2.init();
}

/********************* /
 * Reconfiguracion de los sensores.
 *
 * Al desperta a los sensores, se reinician con la configuracion por defecto.
 * Por ello es necesario reconfigurarlos al volver a tomar medidas.
/********************* /

void reconfig_INAs(){
    Solar.setConversionTime(CONV_TIME_1100);

```

```

Batbu.setConversionTime(CONV_TIME_1100);
Bat1.setConversionTime(CONV_TIME_1100);
Bat2.setConversionTime(CONV_TIME_1100);

Solar.setMeasureMode(CONTINUOUS);
Batbu.setMeasureMode(CONTINUOUS);
Bat1.setMeasureMode(CONTINUOUS);
Bat2.setMeasureMode(CONTINUOUS);

Solar.setResistorRange(0.1, 10);
Batbu.setResistorRange(0.1, 10);
Bat1.setResistorRange(0.1, 10);
Bat2.setResistorRange(0.1, 10);

// Factores de correccion medidos experimentalmente
Solar.setCorrectionFactor(1.0469);
Batbu.setCorrectionFactor(1.0419);
Bat1.setCorrectionFactor(0.9650);
Bat2.setCorrectionFactor(0.9624);

Solar.waitUntilConversionCompleted();
Batbu.waitUntilConversionCompleted();
Bat1.waitUntilConversionCompleted();
Bat2.waitUntilConversionCompleted();
}

/***********************/
* Realiza una medida de todos los sensores.
*
* Inicialmente se verifica si se ha realizado la configuracion inicial de los INAs.
* -Si se ha realizado, se despiertan y reconfiguran
* -En caso contrario se realiza una configuracion inicial antes de la captacion de datos.
* Se realiza 1 medida que es almacenada en la estructura de datos.
* Y finalmente se ponen a bajo consumo los INA.
* @param telemetry Puntero a la estructura de datos que contiene las medidas de tension y
* corriente de las baterias y panel solar.
/***********************/

void measureINA226(telemetry_t *telemetry) {
    // Panel Solar
    if(first_config==false){
        setupINA226Sensors();
        first_config=true;
    }else{
        powerUpINA226();
    }

    reconfig_INAs();

    Solar.readAndClearFlags();
    telemetry->VSolar = Solar.getBusVoltage_V() + (Solar.getShuntVoltage_mV() / 100);
    telemetry->ISolar = Solar.getCurrent_mA();

    // Bateria Backup
    Batbu.readAndClearFlags();
    telemetry->VBatbu = Batbu.getBusVoltage_V() + (Batbu.getShuntVoltage_mV() / 100);
    telemetry->IBatbu = -Batbu.getCurrent_mA();

    // Bateria 1
    Bat1.readAndClearFlags();
    telemetry->VBat1 = Bat1.getBusVoltage_V() + (Bat1.getShuntVoltage_mV() / 100);
    telemetry->IBat1 = -Bat1.getCurrent_mA(); // Invertir signo porque esta al reves
}

```

```

// Bateria 2
Bat2.readAndClearFlags();
telemetry->VBat2 = Bat2.getBusVoltage_V() + (Bat2.getShuntVoltage_mV() / 100);
telemetry->IBat2 = Bat2.getCurrent_mA();
powerDownINA226();
}

/*********************//**
 * Pone en modo de bajo consumo a los INA.
 *
 * Tras una medida no es necesario que se mantengan activos los sensores por lo que es
 * necesario ponerlos a bajo consumo.
*****//

void powerDownINA226() {
    Solar.powerDown();
    Batbu.powerDown();
    Bat1.powerDown();
    Bat2.powerDown();
}

/*********************//**
 * Despierta a los INA del bajo consumo.
 *
 * Al desperta a los sensores, se reinician con la configuracion por defecto.
 * Por ello es necesario reconfigurarlos al volver a tomar medidas.
*****//

void powerUpINA226() {
    Solar.powerUp();
    Batbu.powerUp();
    Bat1.powerUp();
    Bat2.powerUp();
}

```

Fragmento 30: Ficheroina.cpp

```

/**
 * @file ina.hpp
 * @author Estela Mora
 * @author David Andriño
 * @author Hugo Sanchez
 * @author Fernando Sanz
 * @brief Modulo de medida de sensores INA226
 * @version 1.1
 * @date 15-12-2024
 */
#ifndef ina_h
#define ina_h

#include <Wire.h>
#include <INA226_WE.h>
#include "telemetry.hpp"

/** @def I2C_D_PANEL
 * @brief Indica la direccion I2C del INA que toma las medidas del panel solar.
 */
#define I2C_D_PANEL 0x45
/** @def I2C_D_BAT_BU
 * @brief Indica la direccion I2C del INA que toma las medidas de la bateria de BackUp.
 */
#define I2C_D_BAT_BU 0x40
/** @def I2C_D_BAT_1

```

```

@brief Indica la direccion I2C del INA que toma las medidas de la primera bateria.
*/
#define I2C_D_BAT_1 0x44
/** @def I2C_D_BAT_2
@brief Indica la direccion I2C del INA que toma las medidas de la segunda bateria.
*/
#define I2C_D_BAT_2 0x41

/** 
 * @brief Realiza una medida de todos los sensores.
 *
 * Inicialmente se verifica si se ha realizado la configuracion inicial de los INAs.
 * - Si se ha realizado, se despiertan y reconfiguran.
 * - En caso contrario se realiza una configuracion inicial antes de la captacion de datos.
 * Se realiza 1 medida que es almacenada en la estructura de datos.
 * Finalmente se apagan los INA.
 *
 * @param telemetry Puntero a la estructura de datos que contiene las medidas.
 */
void measureINA226(telemetry_t *telemetry);

#endif

```

Fragmento 31: Ficheroina.hpp

```

#include "telemetry.hpp"
#include "hardware.hpp"
#include "ina.hpp"
#include "mqtt_client.hpp"
#include "file_management.hpp"
#include "sleep.hpp"

#define DEBUG

#ifndef DEBUG
#define PRINT_DEBUG(...) Serial.println(__VA_ARGS__)
#else
#define PRINT_DEBUG(...)
#endif

#define ssid          "POC"
#define password      "POWERIoT"

#define MQTT_SERVER   const_cast<char*>("192.168.111.69")
#define MQTT_PORT     4444

#define TIME_SLEEP    2000
#define WIFI_TRIES    35
#define BAT_THRESH    11.65
#define SOLAR_THRESH  20
#define MEASUREMENT_FILE "/measurements.txt"

telemetry_t telemetry = {0, 0, 0, 0, 0, 0, 0, 0, 0};
char        timeStr[50] = "";
struct tm  currentTime;

/** 
 * @brief Intenta realizar la conexion WiFi hasta WIFI_TRIES intentos
 */
static void setup_wifi();

/** 
 * @brief Conmuta los reles en funcion de las medidas tomadas

```

```
*  
* 1. Si la tension del panel solar es mayor que SOLAR_THRESH, se activa el panel solar.  
* 2. Si la tension de la bateria de backup es mayor que BAT_THRESH, se activa la bateria de  
    backup.  
* 3. En caso contrario, se activa la proteccion de sobredescarga.  
*  
* @param telemetry Puntero a la estructura de medidas  
*/  
static void setRelays(telemetry_t *telemetry);  
  
void setup() {  
    pinMode(RELAY_SW_IN, OUTPUT);  
    pinMode(RELAY_SW_OUT, OUTPUT);  
    pinMode(SDA,          OUTPUT);  
    pinMode(SCL,          OUTPUT);  
  
    Serial.begin(9600);  
}  
  
void loop() {  
    measureINA226(&telemetry); // Si no estan los INA conectados, bloquea  
  
    PRINT_DEBUG("[INA] VSolar [V]: " + String(telemetry.VSolar));  
    PRINT_DEBUG("[INA] ISolar [mA]: " + String(telemetry.ISolar));  
    PRINT_DEBUG("[INA] VBatbu [V]: " + String(telemetry.VBatbu));  
    PRINT_DEBUG("[INA] IBatbu [mA]: " + String(telemetry.IBatbu));  
    PRINT_DEBUG("[INA] VBat1 [V]: " + String(telemetry.VBat1));  
    PRINT_DEBUG("[INA] IBat1 [mA]: " + String(telemetry.IBat1));  
    PRINT_DEBUG("[INA] VBat2 [V]: " + String(telemetry.VBat2));  
    PRINT_DEBUG("[INA] IBat2 [mA]: " + String(telemetry.IBat2));  
  
    setRelays(&telemetry);  
  
    setup_wifi();  
  
    if (WiFi.status() == WL_CONNECTED){  
        configTime(3600, 0, "time.nist.gov", "0.pool.ntp.org");  
  
        if (!publishTelemetry(MQTT_SERVER, MQTT_PORT, telemetry)){  
            PRINT_DEBUG("[POWER] ERROR publicando medidas");  
        } else {  
            PRINT_DEBUG("[POWER] Medidas publicadas con exito");  
        }  
  
        getLocalTime(&currentTime, 1000);  
        strftime(timeStr, 50, "[%d/%m/%y - %H:%M:%S]", &currentTime);  
        PRINT_DEBUG("[WIFI] Sincronizado con NTP. Hora obtenida: " + String(timeStr));  
    } else {  
        PRINT_DEBUG("[WIFI] No se ha podido conectar a la red WiFi");  
        snprintf(timeStr, 50, "[00/00/00 - 00:00:00]");  
    }  
  
    if (write_meas(MEASUREMENT_FILE, telemetry, timeStr)){  
        PRINT_DEBUG("[POWER] Fichero escrito con exito");  
    } else {  
        PRINT_DEBUG("[POWER] El fichero no ha podido ser escrito");  
    }  
  
    sleep_low_power(TIME_SLEEP);  
}  
  
static void setup_wifi() {  
    delay(10);
```

```

Serial.print("[WIFI] Intentando conectar a " ssid);

WiFi.begin(ssid, password);
for (int i = 0; i < WIFI_TRIES && WiFi.status() != WL_CONNECTED; i++) {
    delay(500);
    Serial.print(".");
    i++;
}
if(WiFi.status() == WL_CONNECTED){
    PRINT_DEBUG("[WIFI] Conectado a la red WiFi " ssid " con IP: ");
    PRINT_DEBUG(WiFi.localIP());
} else {
    PRINT_DEBUG("[WIFI] No se ha podido conectar a la red WiFi");
}
}

static void setRelays(telemetry_t *telemetry){
    if (telemetry->VSolar > SOLAR_THRESH) {
        PRINT_DEBUG("[RELAY] Panel solar activo");

        digitalWrite(RELAY_SW_IN,LOW);
        digitalWrite(RELAY_SW_OUT,LOW);
    } else if (telemetry->VBatbu > BAT_THRESH){
        PRINT_DEBUG("[RELAY] Bateria backup activa");

        digitalWrite(RELAY_SW_IN,HIGH);
        digitalWrite(RELAY_SW_OUT,HIGH);
    } else {
        PRINT_DEBUG("[RELAY] Activando proteccion sobredescarga. Se detendra el sistema");

        digitalWrite(RELAY_SW_IN,LOW);
        digitalWrite(RELAY_SW_OUT,LOW);
    }
}

```

Fragmento 32: FicheroMAIN_POWER.ino

```

#include "mqtt_client.hpp"

static void reconnect(PubSubClient& client, char* const mqtt_server, int mqtt_port);

bool publishTelemetry(char* const mqtt_server, int mqtt_port, telemetry_t& telemetry){
    WiFiClient wifiClient;
    WiFiUDP ntpUDP;
    PubSubClient client(wifiClient);
    bool statusPublish=false;
    char result[256];

    client.setServer(mqtt_server, mqtt_port);

    if (!client.connected()){
        reconnect(client, mqtt_server, mqtt_port);
    }

    snprintf(result, sizeof(result),
             "[VSolar: %.2f, ISolar: %.2f, VBatbu: %.2f, IBatbu: %.2f, VBat1: %.2f, IBat1: %.2f, VBat2: %.2f, IBat2: %.2f]",
             telemetry.VSolar, telemetry.ISolar,
             telemetry.VBatbu, telemetry.IBatbu,
             telemetry.VBat1, telemetry.IBat1,
             telemetry.VBat2, telemetry.IBat2
    );

    statusPublish = client.publish("v1/devices/me/telemetry", result);
}

```

```

    client.disconnect();
    return statusPublish;
}

void reconnect(PubSubClient& client, char* const mqtt_server, int mqtt_port) {
    for (int i = 0; i < 3 && !client.connected(); i++){

        Serial.print("[MQTT] Iniciando conexion a " + String(mqtt_server) + ":" +
                    String(mqtt_port));

        if (client.connect("Cliente", "Cliente", "Cliente")) {
            Serial.println("[MQTT] Conectado");
        } else {
            Serial.print("[MQTT] Conexion fallida. Estado = " + String(client.state()) + " .
                Reintentando en 0.5 segundos... ");
            delay(500);
        }
    }
}

```

Fragmento 33: Ficheromqtt_client.cpp

```

/**
 * @file mqtt_client.hpp
 * @author David Andriño
 * @author Estela Mora
 * @author Hugo Sanchez
 * @author Fernando Sanz
 * @brief Modulo de cliente MQTT
 * @version 1.1
 * @date 15-12-2024
 */
#ifndef MQTT_CLIENT_HPP
#define MQTT_CLIENT_HPP
#include <WiFiUdp.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <Wire.h>

#include "telemetry.hpp"

/**
 * @brief Numero de reintentos de conexion con el servidor MQTT.
 */
#define REINTENTOS_MQTT 3

/**
 * @brief Crea el cliente MQTT y establece la conexion con el servidor MQTT.
 * Ademas, define la estructura del mensaje para publicar las medidas
 * tomadas y mostrarlas en ThingsBoard. En caso de que ocurra un error
 * en la conexion con el servidor MQTT, se reintentara cada 5 segundos
 * mediante una espera bloqueante.
 * @param mqtt_server Direccion del servidor MQTT.
 * @param mqtt_port Puerto del servidor MQTT.
 * @param telemetry Referencia a una estructura que contiene las medidas de telemetria.
 * @return true si la publicacion de telemetria fue exitosa, false en caso contrario.
 */
extern bool publishTelemetry(char* mqtt_server, int mqtt_port, telemetry_t& telemetry);

#endif

```

Fragmento 34: Ficheromqtt_client.hpp

```
#include "sleep.hpp"

#include <ESP8266WiFi.h>

void sleep_low_power(int time_delay) {
    Serial.println("[SLEEP] Entrando en modo bajo consumo");
    WiFi.mode(WIFI_OFF);
    WiFi.forceSleepBegin();
    system_update_cpu_freq(SYS_CPU_80MHZ);

    delay(time_delay);

    Serial.println("[SLEEP] Saliendo del modo bajo consumo");
    system_update_cpu_freq(SYS_CPU_160MHZ);
    WiFi.forceSleepWake();
    delay(1);
    WiFi.mode(WIFI_STA);

    // No hace falta conectar a la red WiFi, ya que se conectara en el loop
}
```

Fragmento 35: Ficherosleep.cpp

```
/***
 * @file sleep.hpp
 * @author David Andrino
 * @author Estela Mora
 * @author Hugo Sanchez
 * @author Fernando Sanz
 * @brief Modulo de bajo consumo
 * @version 1.1
 * @date 15-12-2024
 */
#ifndef SLEEP_H
#define SLEEP_H

/***
 * @brief Pone al ESP8266 en modo de bajo consumo durante el tiempo que se le indique
 *
 * @param time_delay Tiempo en milisegundos que el ESP8266 estara en modo de bajo consumo
 */
extern void sleep_low_power(int time_delay);

#endif
```

Fragmento 36: Ficherosleep.hpp

```
/***
 * @file telemetry.hpp
 * @author David Andrino
 * @author Estela Mora
 * @author Hugo Sanchez
 * @author Fernando Sanz
 * @brief Definicion de la estructura de telemetria
 * @version 1.1
 * @date 15-12-2024
 */
#ifndef TELEMETRY_H
#define TELEMETRY_H

/***
 * @brief Estructura con las medidas tomadas por los diferentes sensores
```

```
/*
typedef struct {
    float VSolar; /**< Tension panel solar */
    float ISolar; /**< Corriente panel solar */
    float VBatbu; /**< Tension bateria backup */
    float IBatbu; /**< Corriente bateria backup */
    float VBat1; /**< Tension bateria 1 */
    float IBat1; /**< Corriente bateria 1 */
    float VBat2; /**< Tension bateria 2 */
    float IBat2; /**< Corriente bateria 2 */
} telemetry_t;

#endif
```

Fragmento 37: Ficherotelemetry.hpp

Anexo B Perspectivas de la caja

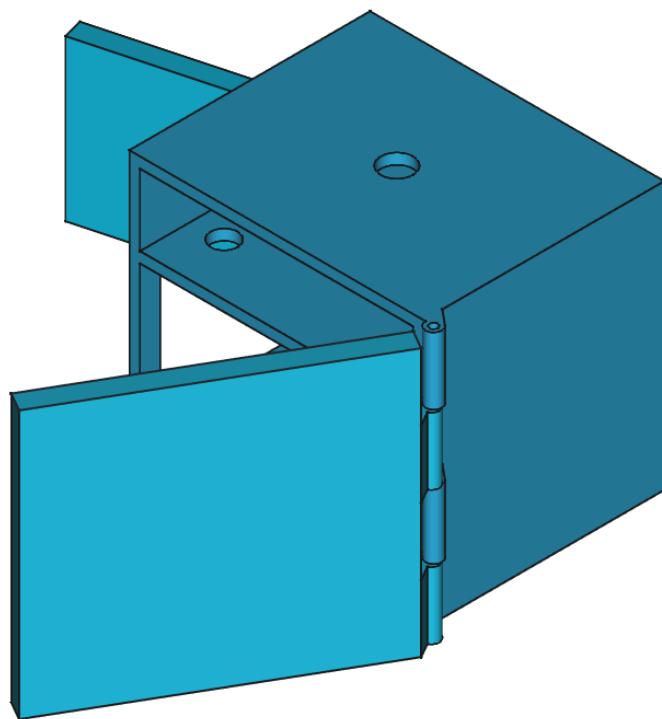


Figura 49: Vista isométrica

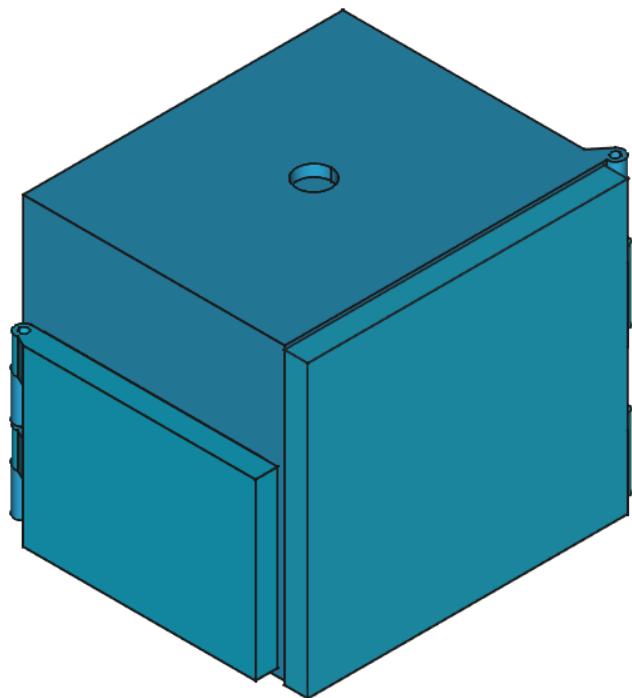


Figura 50: Caja Cerrada

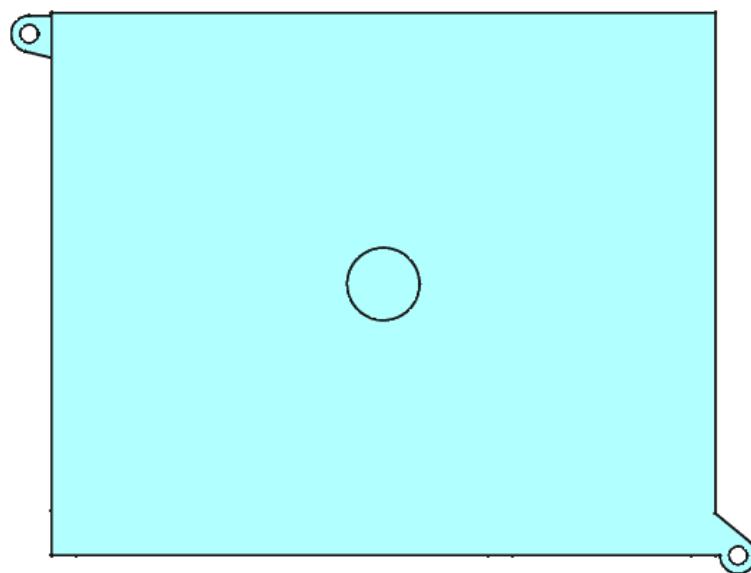


Figura 51: Vista parte superior de la caja

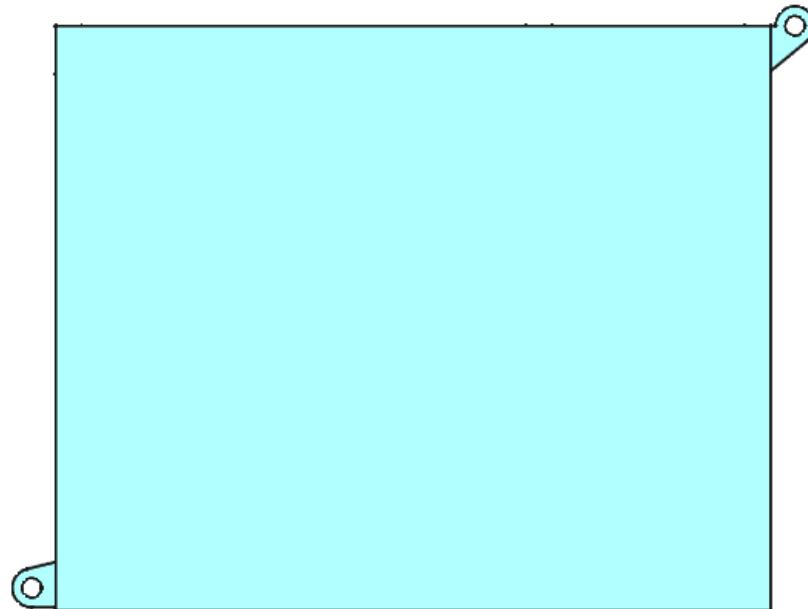


Figura 52: Vista parte inferior de la caja

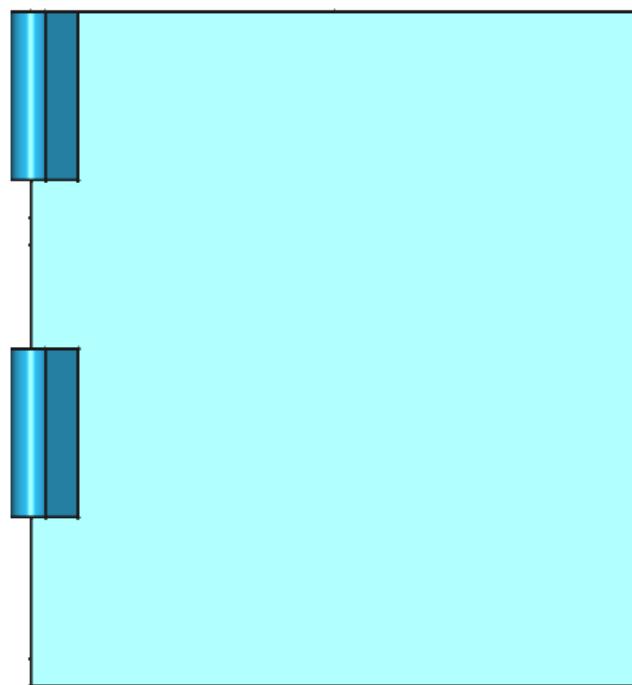


Figura 53: Vista parte derecha de la caja

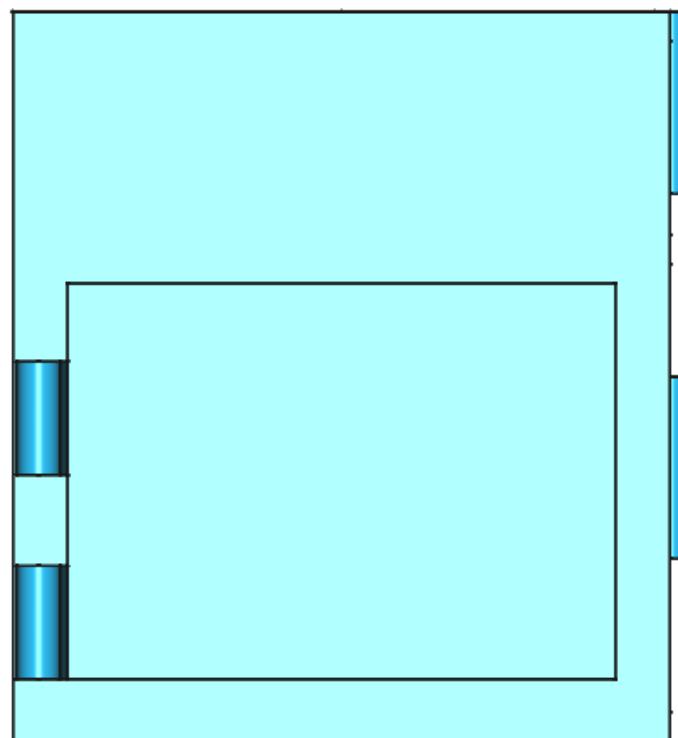


Figura 54: Vista parte izquierda de la caja

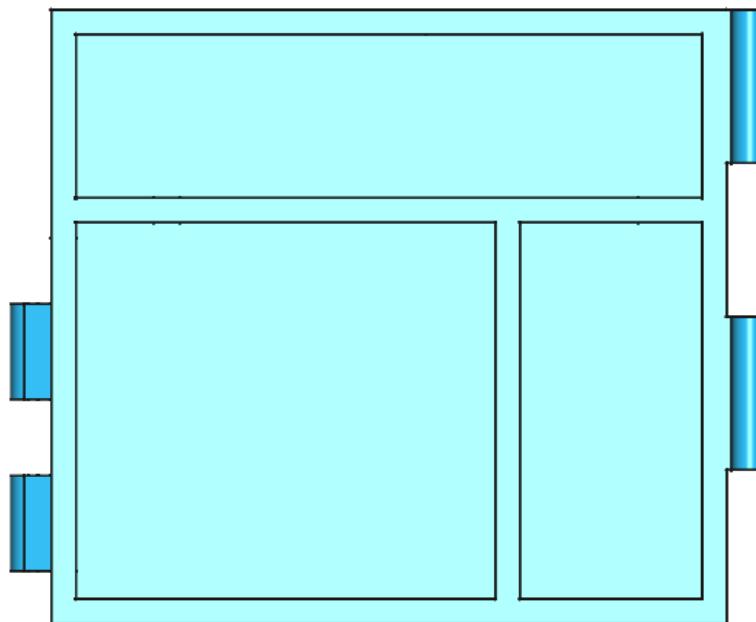


Figura 55: Vista parte frontal de la caja

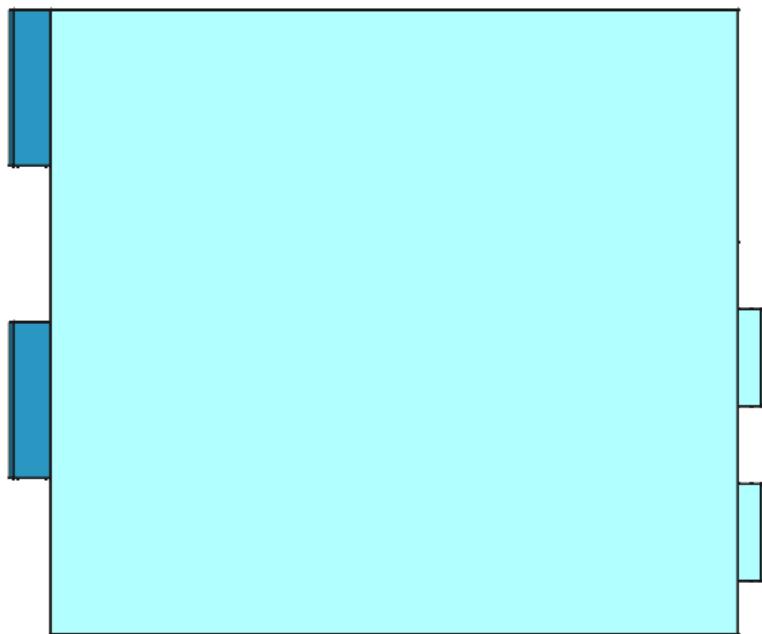


Figura 56: Vista parte de atrás de la caja