

INGENIERÍA DE SISTEMAS ELECTRÓNICOS

Curso 2024-2025. BLOQUE 2

*Título proyecto: InverTech
(Invernadero Inteligente)*

Estudiantes:

Nizar El Azeouzi Amine

Ignacio Sánchez Gil

Álvaro Salvador Ruiz

Aitor Casado de la Fuente



Índice del documento

1	ESPECIFICACIONES.....	2
1.1	Especificaciones iniciales del sistema a diseñar y construir.....	2
1.2	Especificaciones finales del sistema diseñado y construido.	8
2	DESARROLLO DE SUBSISTEMAS.....	11
2.1	Alimentación	11
2.2	Circuitos analógicos.....	12
2.2.1	Acondicionador RTD.....	12
2.2.2.	Acondicionador para el consumo	18
2.3	Sensores integrados (I2C, SPI, etc).....	21
3	ESQUEMA ELECTRÓNICO COMPLETO.....	26
4	SOFTWARE.	28
4.1	Interfaz de usuario.	28
4.2	Descripción de cada uno de los módulos del sistema.....	30
5	DEPURACION Y TEST.	69
5.1	Pruebas realizadas software.	69
MÓDULO RFID	69	
MÓDULO LCD	70	
MEMORIA EEPROM AT24C256	70	
SENSOR BME680.....	71	
SENSOR HC-SR04.....	71	
SENSOR BH1750	73	
COMUNICACIÓN ENTRE PLACAS – UART	74	
PRUEBAS CON TODOS LOS MÓDULOS - UART.....	75	
FUNCIONAMIENTO DE LA PÁGINA WEB	79	
FUNCIONAMIENTO DEL BAJO CONSUMO	79	
5.2	Pruebas realizadas hardware.	82
5.2.1	Acondicionador de temperatura	82
5.2.2	Acondicionador para el consumo.....	84
6	CONCLUSIONES.	87
7	PRESUPUESTO FINAL.....	88
8	EQUIPO DE TRABAJO.....	89
9	ACRÓNIMOS UTILIZADOS.....	90
10	BIBLIOGRAFÍA UTILIZADA.....	91

1 ESPECIFICACIONES.

1.1 Especificaciones iniciales del sistema a diseñar y construir.

1.1.1. Descripción de la idea

El objetivo de este proyecto es diseñar un sistema de control automatizado para la monitorización y optimización de un invernadero, permitiendo así el seguimiento remoto de las condiciones ambientales más importantes del mismo. Además, se pretende mostrar dichos parámetros en un servidor web de forma que el usuario pueda tener controlado el invernadero en cuestión. Dicho sistema contará con múltiples sensores y actuadores que permiten medir los parámetros físicos claves de un invernadero, tales como temperatura, humedad, luminosidad, calidad del aire, capacidad del tanque de agua para el riego, de forma que permite analizar las variables críticas para el crecimiento de los cultivos. Entre la conexión oportuna de dos tarjetas núcleo STM32F429ZI (maestro y esclavo), se podrá observar, tanto en el LCD como en el servidor web los parámetros clave anteriormente descritos. Además, contará con dos modos principales de funcionamiento: **PRESENCIAL** y **REMOTO**. El modo **PRESENCIAL** simulará cuando el usuario se encuentra presente en el invernadero en sí. La forma de acceder será mediante un lector RFID, para que, en el momento que se detecte la correcta identificación se podrá “acceder” al invernadero y comprobar los parámetros. La visualización será en el LCD de la tarjeta deMBED de aplicaciones. En pequeños intervalos de tiempo de 10 segundos se podrán observar en diferentes ventanas el consumo actual, luminosidad, nivel de agua, calidad del aire, temperatura y humedad. La forma de salir de este modo es volviendo a pasar por el lector RFID la identificación, simulando la salida presencial del invernadero y entrando en bajo consumo. El segundo modo de funcionamiento, el modo **REMOTO**, simula el usuario que se encuentra en una ubicación distinta a la del invernadero y puede acceder de forma remota gracias al servidor web. En este servidor se podrán observar los mismos parámetros además del consumo. Contará con botones para cuando se desee salir del invernadero, introducir a la tarjeta en el modo de bajo consumo. Por último, periódicamente gracias al protocolo SNTP que implementará la tarjeta máster, se despertará del bajo consumo a la tarjeta esclava para hacer medidas y guardarlas en la memoria no volátil (EEPROM), para posteriormente volver a dormir a la tarjeta. Todas estas medidas almacenadas se verán en una ‘subweb’ de la página web.

1.1.2. Descripción detallada. Bloques.

1.1.2.1. Digital

Como se mencionó en el punto anterior, la aplicación contará con dos tarjetas núcleo, una funcionará como maestro y la otra como esclavo. La **tarjeta maestra** será la encargada de procesar toda la información, además de implementar la gestión de las dos formas de acceso al invernadero. Contará con los siguientes bloques, detallados en la propuesta inicial del bloque 2:

- **Memoria EEPROM:** dicha memoria implementada mediante el protocolo I2C guardará los parámetros del invernadero en páginas de 64 bytes. Además, en la misma se guardará la hora a la que se realizaron dichas medidas para tener una marca temporal. Dicha memoria es la **EEPROM AT24C256** que cuenta con una capacidad de 256 Kbit.
- **LCD:** será el que implementa la tarjeta de aplicaciones MBED en el cual se mostrará los parámetros durante el modo PRESENCIAL, además de servir como interfaz para el usuario. Esta emplea el protocolo **SPI**.
- **RFID:** con el objeto de poder tener un control de acceso al invernadero (medida de seguridad) y hacerlo más privado como si simulase que es único del usuario, dispondremos de este lector que emplea el protocolo SPI. Dicho sensor RFID simulará la entrada/salida del invernadero.
- **Altavoz/Zumbador:** a fin de saber que la lectura de la tarjeta RFID ha sido correcta y exitosa, se emitirá un pequeño pitido procedente del zumbador de la tarjeta MBED de aplicaciones, emitiendo esa onda de 1 kHz mediante un GPIO que implementará una onda PWM.
- **Led RGB:** este led realizará dos funciones. Servirá tanto para indicar si la identificación del sensor RFID ha sido correcta o incorrecta, además de indicar el nivel de agua del tanque en el modo PRESENCIAL. Los indicadores serán: verde (identificación correcta y nivel del tanque de agua superior al 70%); amarillo (nivel del tanque de agua entre 20% y 70%); por último, el rojo (identificación incorrecta y nivel del tanque de agua por debajo del 20%).

Por otra parte, la tarjeta esclava que será la encargada de realizar toda la adquisición de los parámetros del invernadero y enviarlos a la maestra, dispondrá de los siguientes bloques:

- **BME680:** este sensor, que funcionará mediante el protocolo **I2C**, será encargado de captar la **humedad** y **calidad del aire** presentes en el invernadero.
- **HC-SR04:** este sensor se encargará de simular la **cantidad o nivel de agua** que hay presente en el tanque de agua que dispondrá el invernadero para regar los cultivos. Para ello, este sensor que es ultrasónico emitirá ráfagas de 40 kHz que rebotarán en el agua, permitiendo calcular la distancia. Para ello tiene dos pines **GPIO**: uno que emite los pulsos y otro que recibe la onda rebatida, permitiendo calcular la distancia.
- **BH1750:** este sensor se encargará de detectar la **luminosidad** presente en el invernadero, proporcionando un feedback sobre el tiempo presente al estar de forma remota. Para ello, la luminosidad se devolverá mediante el protocolo de comunicaciones **I2C**.

1.1.2.2. Analógico

De la misma forma que se detalló en la propuesta del proyecto, se dispondrán de dos subsistemas analógicos, es decir dos PCBs: una dedicada a la **alimentación independiente** y al **acondicionador del consumo** de la tarjeta esclava; y otra para la acondicionar la tensión que proporciona la RTD para obtener la **temperatura ambiente**.

1. Alimentación + Consumo: a fin de proporcionar una tensión estable al microcontrolador y al acondicionador de la RTD, se ha diseñado una fuente de alimentación basada en baterías recargables y reguladores lineales. La batería estará compuesta de tres baterías 18650 de 3000 mAh recargables de 3,7V nominales conectadas en serie, proporcionando una tensión total de 11,1 V al circuito.

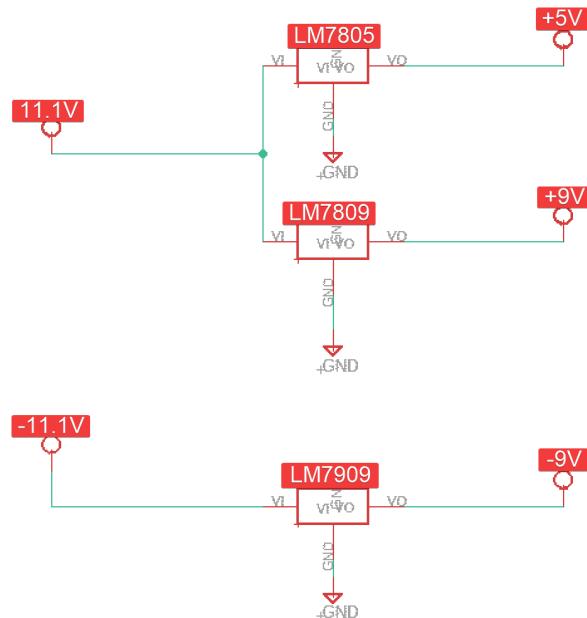
I. Regulación de la parte positiva:

- Los 11.1V de las baterías pasarán por un regulador lineal LM7805, cuya salida será de 5V, destinada a alimentar el microcontrolador.
- En paralelo, los 11.1V también pasarán por un regulador lineal LM7809, obteniendo 9V, que se utilizarán para alimentar los amplificadores operacionales del sistema de consumo.

II. Regulación de la parte negativa:

- Los -11.1V generados por las tres baterías conectadas en serie se estabilizarán con un regulador lineal LM7909, cuya salida será de -9V, necesarios para la alimentación de los amplificadores operacionales del sistema de consumo.

Además, la salida de 5V obtenida después del LM7805 se dirigirá hacia la resistencia shunt, donde se encuentra el sistema de consumo, y posteriormente alimentará al microcontrolador.



Para conocer el **consumo** del microcontrolador, se implementará un sistema de medición basado en una resistencia shunt de bajo valor óhmico y un amplificador de instrumentación diseñado con tres amplificadores operacionales OP07.

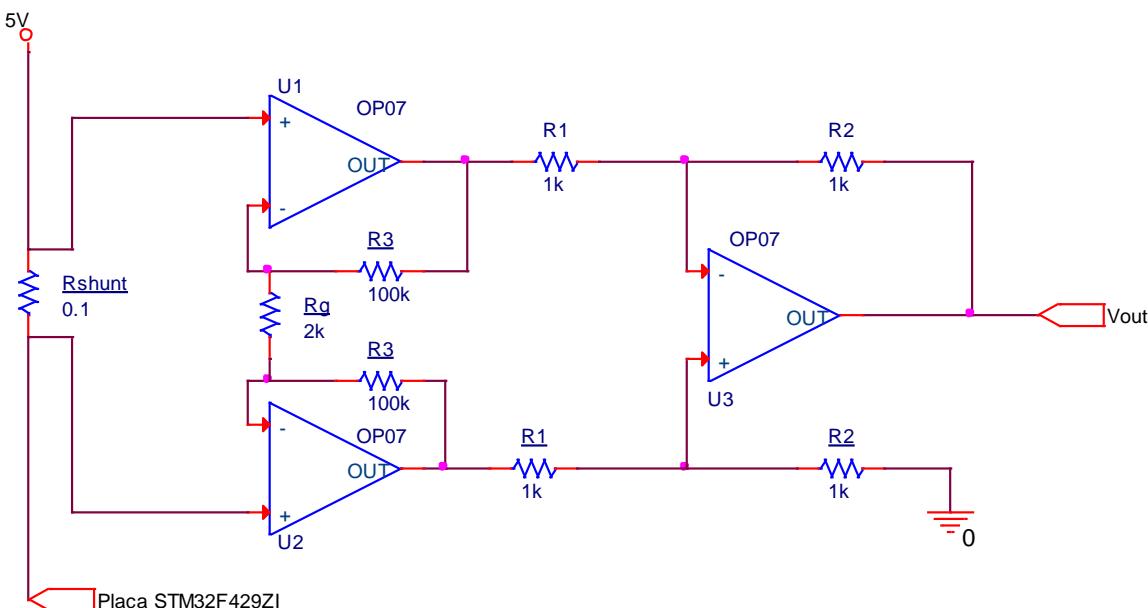
La resistencia shunt se colocará en serie con la línea de alimentación de 5V, que proviene de la salida del convertidor DC-DC reductor. Su valor será bajo para minimizar la pérdida de energía en el sistema y reducir su impacto en la alimentación del microcontrolador. Cuando la corriente fluye a través de la resistencia shunt, se genera una pequeña caída de tensión proporcional a la corriente consumida por el microcontrolador. Esta diferencia de potencial es capturada por el amplificador de instrumentación para ser amplificada y posteriormente leída por el microcontrolador.

La salida del amplificador de instrumentación amplificada en un factor de 100 se conectará a una entrada analógica del microcontrolador. A partir de esta lectura, se realizará el cálculo del consumo en la placa mediante la relación:

$$I_{consumo} = \frac{V_{shunt}}{R_{shunt}} = \frac{\frac{V_{adc}}{100}}{0.1}$$

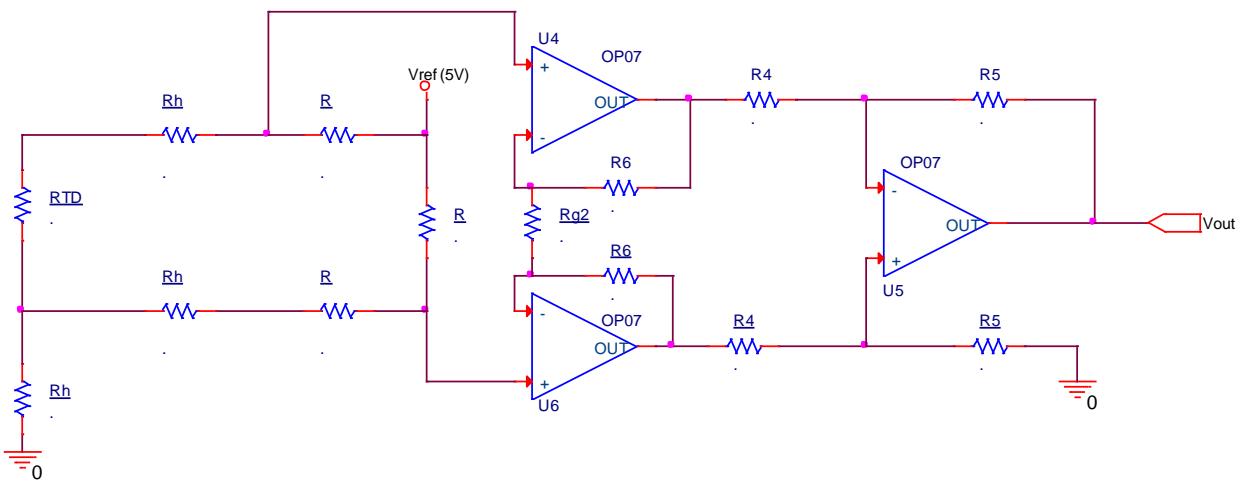
Donde:

- V_{shunt} es la caída de tensión en la resistencia shunt.
- $R_{shunt} = 0.1$ ohm es el valor de la resistencia.
- V_{ADC} es la lectura obtenida por el microcontrolador tras la amplificación.



2. Acondicionador de temperatura

Para poder obtener el dato real de la temperatura, se va a diseñar un sensor de temperatura basado en una RTD, concretamente una Pt-100, que permite variar su valor óhmico entre 0 y $100\ \Omega$. Se trata de un componente pasivo, cuyo funcionamiento se basa en el principio de que la resistencia eléctrica de un material conductor cambia de manera predecible con la temperatura. En el interior, dispone de un alambre de platino, caracterizado por su precisión, estabilidad y amplio rango de temperatura. Como se ha dicho antes, la variación del sensor es de 0 a $100\ \Omega$ para temperaturas de -5 °C a 45 °C, y para nuestra aplicación se ha optado por diseñar un puente de Wheatstone con una rama activa, conectando la RTD a 3 hilos, minimizando así el error por los cables. El valor de resistencias todavía no ha sido elegido, pero será el necesario para dar una tensión diferencial de 0 a $\approx 50\ \text{mV}$. Para poder adaptar este margen, se empleará un amplificador de instrumentación, empleando amplificadores operacionales y con la ganancia necesaria de esta etapa, consiguiendo a la salida un margen dinámico de 0 a 3.3 V, adaptado al margen que pueden leer los ADCs de la tarjeta. Dicho amplificador de instrumentación se compondrá de dos etapas: etapa 1 y etapa 2. La primera tapa está formada por dos amplificadores operacionales en topologías inversoras y no inversoras, mientras que la segunda se compone de un amplificador diferencial. Este tipo de configuración proporciona numerosas mejores primordiales en la aplicación, eliminando el error y el offset gracias al CMRR y a la alta impedancia existente a la entrada. El parámetro CMRR, como su propio nombre indica (Common Mode Rejection Ratio), permite rechazar señales presentes en ambas entradas del AI. Para ello, a la primera etapa del AI se le da toda la ganancia, mientras que la segunda etapa tiene valor unitario. Esto permite amplificar la señal deseada y eliminar esa señal indeseada presente en el modo común. El esquemático es el siguiente:



1.1.2.3. Comunicación entre placas

La comunicación entre dos placas STM32F429ZI se realizará mediante el protocolo SPI, en el que una placa actuará como maestro y la otra como esclavo.

- Modo de operación: SPI en modo Full-Duplex
- Frecuencia de reloj: Configurada según la velocidad de operación deseada, garantizando estabilidad en la transmisión.
- Orden de bits: MSB primero.
- Polaridad y fase del reloj (CPOL, CPHA): por defecto, modo 0: CPOL=0, CPHA=0.
- Tamaño de datos: 8 bits por transferencia.
- Selección de esclavo (NSS): Controlado manualmente mediante un pin GPIO.

I. Asignación de pines:

- MOSI: PA7 (maestro) conectado a PA7 (esclavo).
- MISO: PA6 (maestro) conectado a PA6 (esclavo).
- SCK: PA5 (maestro) conectado a PA5 (esclavo).
- NSS: Configurado manualmente en un GPIO.

II. Proceso de comunicación:

1. El maestro activa la comunicación bajando el pin NSS (PA4) a nivel bajo.
2. Se envía un byte de comando para solicitar una lectura de datos.
3. El esclavo responde con los datos correspondientes en los bytes siguientes.
4. Una vez terminada la transmisión, el maestro pone NSS en alto para finalizar la comunicación.

III. Comandos a utilizar:

Para la comunicación entre maestro y esclavo, se definirá un conjunto de comandos de un solo byte, que el maestro enviará para solicitar la lectura de distintas variables. Cada comando estará seguido de los bytes correspondientes a la respuesta del esclavo.

Comando	Descripción	Respuesta del esclavo
0x01	Leer temperatura	2 bytes (valor en grados)
0x02	Leer humedad	2 bytes (valor en %)
0x03	Leer calidad aire	2 bytes (valor numérico)
0x04	Leer luminosidad	2 bytes (valor numérico)
0x05	Leer cantidad de agua	2 bytes (valor en ml)
0x06	Leer estado de batería	2 bytes (voltaje en mV)
0x07	Escribir la hora	3 bytes (hora, minutos y segundos 1 byte cada uno)
0x08	Escribir la fecha	4 bytes (días y meses 1 byte, año 2 bytes)

Cada respuesta de los sensores está compuesta por 2 bytes, donde el primer byte corresponde a la parte alta del valor y el segundo a la parte baja, permitiendo una representación precisa de los datos.

IV. Manejo del NSS en la comunicación:

Dado que solo hay un esclavo en la comunicación, se utilizará un pin GPIO para manejar manualmente la selección del esclavo (NSS). En cada transmisión:

1. Se pone el pin NSS en bajo antes de enviar un comando.
2. Se envía el byte de comando.
3. Se espera la respuesta del esclavo y se leen los bytes recibidos.
4. Se pone el pin NSS en alto al finalizar la comunicación.

1.2 Especificaciones finales del sistema diseñado y construido.

Tras ir desarrollando el proyecto, el grupo se ha encontrado con retos que han obligado a cambiar algunas funcionalidades. Debido a ello, en esta sección se van a explicar y detallar aquellos bloques que se han cambiado con respecto a las especificaciones finales además de dar una ligera explicación de que ha motivado al cambio:

1. **Comunicación:** tras numerosas pruebas, no se pudo conseguir exitosamente la comunicación **SPI** entre las dos tarjetas núcleo. Por lo tanto, se decidió cambiar dicha comunicación a **UART**, ya que se consideró que su implementación era más sencilla. Aun así, se contaron con problemas como la sincronización entre las dos tarjetas núcleo con los respectivos hilos dedicados a la comunicación. La secuencia de comandos que se ha seguido en el proyecto es la siguiente:

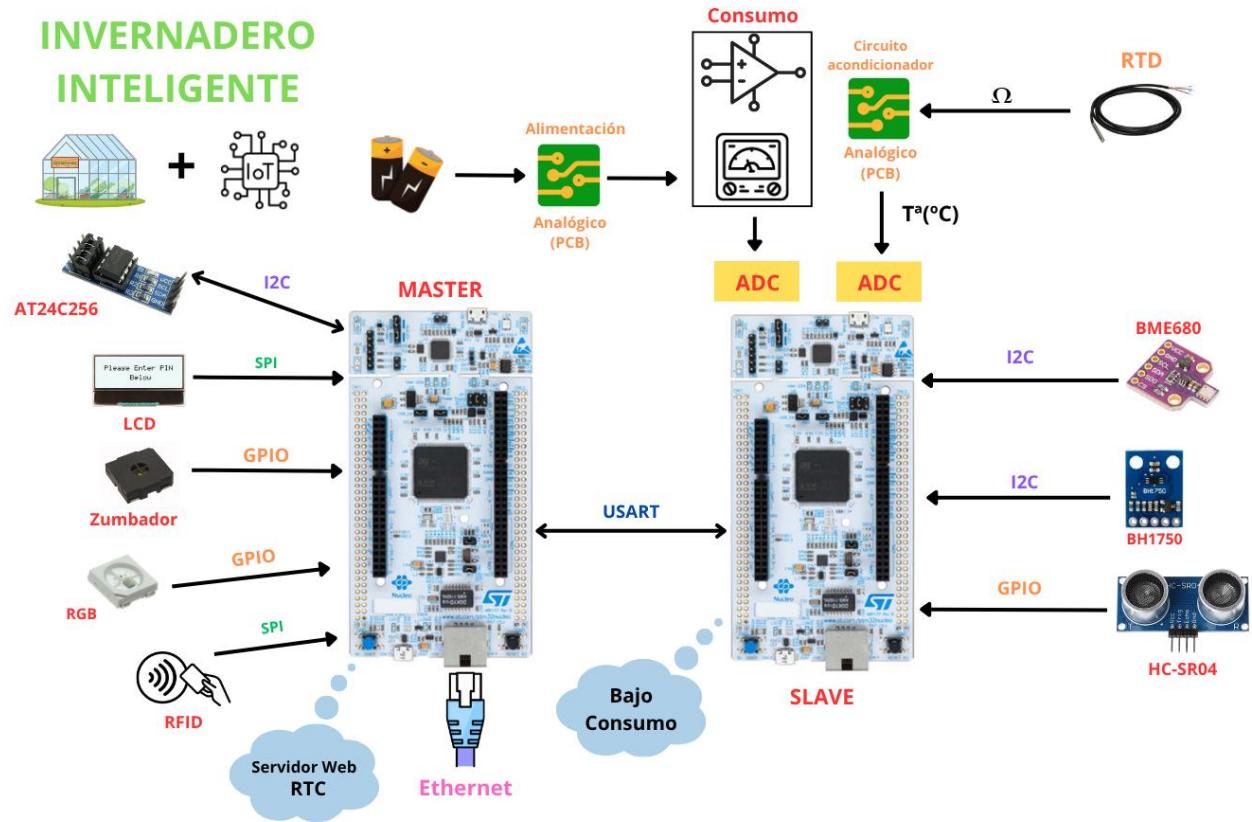
Tipo de Trama	Estructura	Bytes	Contenido	Descripción
◆ Comando Entrante (RX)	[0x7E] [CMD] [0x7F]	3 bytes	CMD: código de comando enviado por el máster	Instrucción que recibe el slave para ejecutar una acción
◆ Trama Inválida (RX)	[0xFF] [0x00] [0x7F]	3 bytes	Todos los bytes intermedios a cero	Indica una trama mal formada; se ignora o descarta
◆ Respuesta de Sensor (TX)	[0x7E] [D1] [D2] ... [D12] [0x7F]	14 bytes	D1-D12: datos sensorizados	Enviada al máster con los datos adquiridos por los sensores
◆ Respuesta Sleep/Wake (TX)	[0x7E] [0x00] ... [0xAA] [0x7F]	14 bytes	0xAA: modo sleep/wake, seguido de ceros	Confirma entrada/salida de modo bajo consumo
◆ Trama de Error	[0xFF] [0x00] ... [0x00] [0x7F]	14 bytes	0xFF: código de error, seguido de ceros	para indicar fallo en ejecución o comando no reconocido

2. **Alimentación:** a fin de reducir el presupuesto y conseguir una parte de alimentación algo más sencilla, se eliminaron varios elementos. La alimentación negativa desde las baterías que se pensó (-11,1 V) se ha eliminado dado a que no era viable ya que se necesitaría el doble de pilas/baterías. Se ha decidido eliminar el regulador que proporciona salida negativa (**LM7909**) ya que se va a partir de 11,1 V proporcionados por tres pilas de 3,7 V en serie y sustituirlo por un inversor **LMC7660**, que invierte la tensión de entrada, es decir que se obtienen a la salida los -9 V necesarios para alimentar a los amplificadores operacionales del sistema. Además, se ha eliminado el regulador que proporciona 5 V (**LM7805**) ya que se ha decidido utilizar la alimentación de 5 V que proporciona la tarjeta núcleo. Se han valorado problemas de referencias, pero como posteriormente todo estará unido, no habrá ningún tipo de problema.

Una vez detallada cada corrección de la correspondiente especificación se van a recoger en una serie de puntos las especificaciones finales del proyecto:

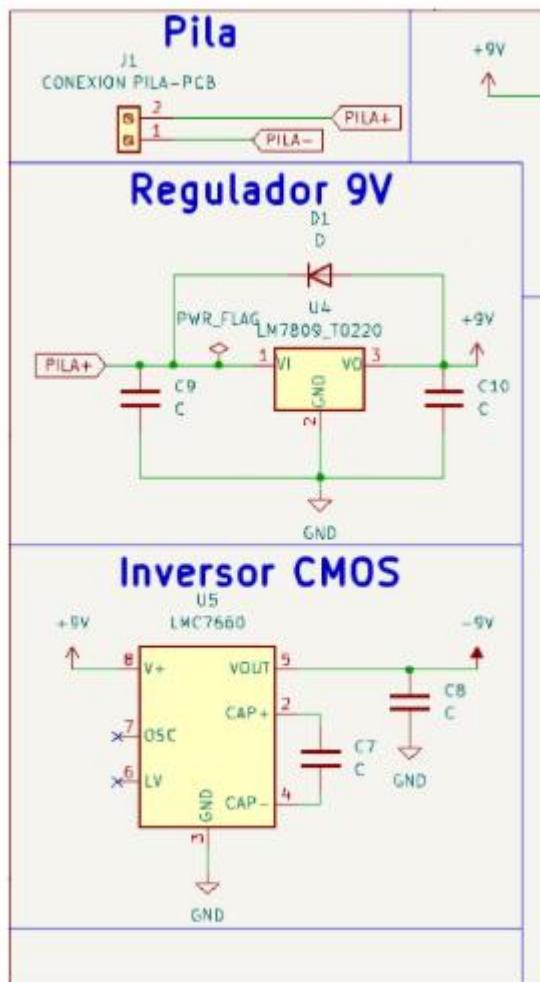
- **Tarjeta maestra y esclava:** se usarán las empleadas en la asignatura, **STM32F429ZI**.
- **Conexión con el servidor:** cable Ethernet a la tarjeta maestra. La hora y fecha del sistema se implementa con el SNTP y RTC.
- **Alimentación tarjeta maestra:** cable micro USB.
- **Alimentación tarjeta esclava:** +9 V por el pin ‘Vin’ de dicha tarjeta, generados desde un circuito de alimentación autónomo compuesto por reguladores.
- **Dos PCB’s:** circuito que integra en uno solo la parte de alimentación y el acondicionador para el consumo del sistema; y por otra parte el acondicionador para una RTD el cual acondicionará la temperatura ambiente.
- **Módulos analógicos diseñados por los estudiantes:** los estudiantes han diseñado y caracterizado dos amplificadores de instrumentación. El acondicionador para el consumo que tiene colocado una R_{shunt} de $0,1 \Omega$ a la entrada de este; después, el acondicionador de temperatura, el cual tiene un puente de Wheatstone a la entrada del mismo con una Pt-100.
- **Comunicación entre tarjetas:** se emplea el protocolo **USART** como se detalló anteriormente, conectando tarjeta maestra (TX, RX) y tarjeta esclava (RX, TX) en dicho orden, además de tener un punto de referencia (GND) común entre las dos tarjetas.
- **Memoria no volátil:** se emplea una tarjeta **EEPROM AT24C256** de 256 Kbit de memoria, comunicando con la tarjeta máster mediante el protocolo I2C y almacenando 10 medidas de los parámetros característicos más importantes del invernadero.
- **Periféricos I2C:** contaremos con dos sensores que estarán conectados a la tarjeta esclava. El primero, BME680 que es el encargado de medir la humedad y la calidad del aire; después tenemos el sensor BH1750 el cual medirá la luminosidad del invernadero.
- **Periféricos SPI:** estos dos periféricos están conectados a la tarjeta maestra. El primero es el **LCD** de 128x32 px presente en la tarjeta mbed de aplicaciones. En este se mostrarán los parámetros más significativos del sistema durante el modo **PRESENCIAL**. Por otra parte, tenemos el sensor **RFID**, el cual servirá para ‘acceder’ al invernadero en dicho modo, detectando únicamente los identificadores guardados en el sistema.
- **Periféricos GPIO:** el primer módulo es el sensor **HC-SR04** con el que se medirá el nivel de agua del invernadero. Los otros dos módulos están presentes en la tarjeta mbed de aplicaciones: el primero es el **led RGB** mediante el cual se mostrará si el acceso al invernadero ha sido correcto o erróneo además de indicar el nivel de agua en el depósito; el segundo es el **altavoz**, mediante el cual indica con un pitido de 1 kHz si se ha podido acceder al invernadero o no.
- **Bajo consumo:** en el sistema se emplea el **Sleep Mode** en la tarjeta esclava. Este modo se activa/desactiva mediante un pin configurado como interrupción, el PB1, simulando un **Chip Select**. A nivel alto, está desactivado, y a nivel bajo el modo sleep está activado.

El diagrama de bloques mostrado incluye, de forma ilustrada, todos los elementos que componen el sistema desarrollado por los estudiantes, en el cual se especifica cada línea de conexión. Posteriormente se mostrará un esquema eléctrico completo con cada conexión a ambas tarjetas núcleo:



2 DESARROLLO DE SUBSISTEMAS.

2.1 Alimentación



El circuito mostrado está diseñado para proporcionar la alimentación necesaria a una placa SLAVE y a una placa adicional que incluye un sensor de temperatura. Para ello, se generan dos tensiones distintas: una tensión positiva de +9 V y una tensión negativa de -9 V, lo que permite disponer de alimentación simétrica para los componentes que así lo requieren.

La alimentación del sistema proviene de una pila conectada a través del conector J1. Esta pila entrega una tensión de aproximadamente 11 V. Este valor ha sido elegido para asegurar que se supera el voltaje mínimo de entrada (dropout) del regulador lineal LM7809, permitiendo así una regulación estable de salida a 9 V. A continuación, la tensión de la pila pasa por un diodo (D1) que protege el circuito ante una posible inversión de polaridad. Después, entra en el regulador lineal LM7809 (U4), que reduce la tensión de entrada a un valor fijo de +9 V. Para estabilizar tanto la entrada como la salida del regulador, se emplean los condensadores C9 y C10. Esta tensión de 9 V generada por el regulador es utilizada como fuente de alimentación principal para la placa SLAVE. Paralelamente, el circuito incluye un inversor de tensión basado en el integrado LMC7660 (U5). Este componente toma la salida de +9 V proporcionada por el LM7809 y la convierte en -9 V. Para su correcto funcionamiento, el LMC7660 requiere condensadores auxiliares (C7 y C8) que permiten realizar las fases de carga y descarga del sistema de inversión. La tensión -9 V obtenida se emplea para alimentar el sensor de temperatura, el cual necesita una alimentación simétrica ± 9 V para funcionar correctamente.

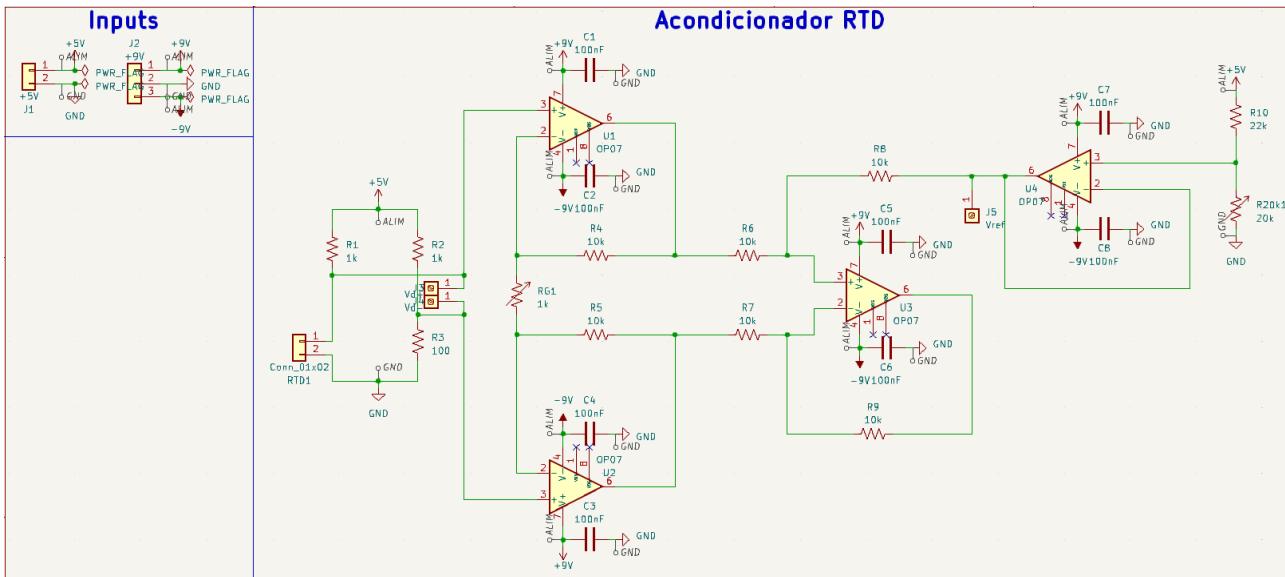
En resumen, el circuito transforma la tensión de una pila de 11 V en una alimentación dual: +9 V para la placa SLAVE y ± 9 V (simétrica) para la placa del sensor de temperatura. Esto permite que ambos sistemas funcionen con la tensión adecuada a partir de una única fuente de alimentación portátil.

Componente	Función	Tensión
Pila	Fuente primaria	~11V
LM7809	Regulador lineal	+9V
LM7660	Inversor CMOS	-9V
Placa SLAVE	Alimentación directa	+9V
Sensor temperatura	Alimentación simétrica	± 9 V

2.2 Circuitos analógicos

2.2.1 Acondicionador RTD

- Esquema Eléctrico:



- Diseño y caracterización:

Como se ha comentado anteriormente, el objetivo es obtener una temperatura a partir de la variación de tensión en una RTD (Pt-100). Para ello, se opta por el circuito comúnmente conocido como puente de Wheatstone, que consta de la diferencia entre dos voltajes obtenidos a raíz de dos divisores resistivos.

En lo que al cálculo de la tensión diferencial en el puente se obtiene a raíz de dos divisores resistivos, donde los valores resistivos son escogidos de tal forma que el puente este correctamente balanceado, por ejemplo, $R_1 = R_2 = 10R$ y $R_3 = R$ en donde $R = 100 \Omega$. Dado a que el objetivo del acondicionador es trabajar entre -5°C y 45°C , se obtiene la siguiente tensión diferencial:

$$V_d = V_1 - V_2 = V_{ref} \cdot \left(\frac{RTD}{10R + RTD} - \frac{R}{10R + R} \right) \rightarrow 98,075\Omega \leq RTD \leq 117,325\Omega$$

$$V_d = \begin{cases} RTD(-5^\circ\text{C}) = 98,075\Omega \rightarrow V_d = -7,96 \text{ mV} \\ RTD = 117,325\Omega \rightarrow V_d = 70,5 \text{ mV} \end{cases}$$

Por otra parte, el amplificador de instrumentación internamente se divide en dos etapas. La primera etapa está formada por dos amplificadores, un inversor y un no inversor, mientras que la segunda es un amplificador diferencial. Esta configuración hace que el rendimiento del amplificador mejore. La etapa 1 da toda la ganancia del amplificador mientras que la etapa 2 se caracteriza por tener una ganancia unitaria. El objetivo del amplificador es tener la mínima ganancia común posible y acumular toda la ganancia posible en la primera etapa:

$$Ad_1 = \left(1 + \frac{2 \cdot R_3}{R_G} \right); Ac_1 = 1; Ad_2 = \frac{R_2}{R_1}; Ac_2 = 0$$

Por lo tanto, si nos fijamos en el parámetro CMRR que es la capacidad que tiene el amplificador para rechazar señales de modo común, es decir señales en ambas entradas de dicho amplificador. Por lo tanto, para lograr un mayor CMRR, se ha de asignar la mayor ganancia a la primera etapa del AI ya que la segunda etapa tiene un valor unitario. Esto permite amplificar la señal diferencial deseada y atenuar la señal en modo común no deseada:

$$CMR_{AI} = \frac{A_D}{A_C} = \frac{\frac{A_{D1} \cdot A_{D2}}{A_{D1} \cdot A_{D2}}}{\frac{CMR_1 \cdot CMR_2}{CMR_1 \cdot CMR_2}} = CMR_1 \cdot CMR_2$$

Por lo tanto, nos llevamos toda la ganancia a la primera etapa. Dado a que el objetivo es tener una tensión a la salida del AI que varíe entre 0 y 3.3 V para que la pueda soportar el ADC. Por lo tanto, la ganancia del AI será de:

$$V_o = A_{AI} \cdot V_d = \left(1 + \frac{2 \cdot R_3}{R_G}\right) \cdot V_d \rightarrow \begin{cases} V_o = 0 \text{ V} \rightarrow V_d = -7,96 \text{ mV} \\ V_o = 3.3 \text{ V} \rightarrow V_d = 70,5 \text{ mV} \end{cases}$$

$$A_{d1} = \frac{3,3 - 0}{70,5 \text{ mV} + 7,96 \text{ mV}} = 42 \text{ V/V}$$

El último circuito que aparece es el ajuste de offset, compuesto por un seguir de tensión que da una tensión dependiendo de la variación del potenciómetro que lo compone. Esto será necesario ya que habrá que ajustar el valor de salida al valor deseado.

Una vez montado el circuito procedemos a realizar el ajuste de este acondicionador. Para ello, nuestro objetivo es conseguir sacar como hemos dicho unos valores de tensión a la salida entre 0 y 3.3 V para unos valores de temperatura de -5 °C a 45 °C. Esto se conoce como sensibilidad del acondicionador, la cual puede darse entre V/Ω o V/°C:

Vout (V)	T (°C)	RTD	Sensibilidad
3,3	45	117,325	0,171428571 V/Ω
0	-5	98,075	0,066 V/°C

Por lo tanto, la sensibilidad a la que hemos de ajustar este acondicionador es la que aparece en el cuadrado amarillo. Nuestro acondicionador se va a caracterizar mediante una recta: $y = mx + n$, donde la pendiente (m) es la sensibilidad/ganancia indicada en V/Ω. La variable independiente (x) es la RTD, la cual dependiendo del valor óhmico tomará un valor de tensión dependiendo de la temperatura. Por último, la (n) será el offset.

Lo primero a realizar será obtener esta sensibilidad, por lo que mediante dos resistencias lo suficientemente alejadas iremos variando el potenciómetro R_{G1} hasta conseguir esa sensibilidad:

RTD	T	Vout
120,01	51,974026	3,6212
99,688	-0,81038961	0,14065
Sensibilidad		0,17127005 (m)

Esas resistencias han sido medidas a cuatro hilos para conseguir una mejor precisión. Ahora mismo no es necesario fijarnos en el valor de salida ya que se va a corregir con el circuito corrector de offset, ya que dicha corrección no afecta a la ganancia. Para ello, teóricamente para la resistencia de -5 °C que es para la que debería de dar 0 V, vamos a obtener la (n) teórica de nuestra recta:

$$V_o = 0 \text{ V} = 0,1714 \cdot RTD(-5 \text{ °C}) + n \rightarrow n = -0,1714 \cdot 98,075 = -16,8$$

Por lo tanto, para la resistencia real medida multiplicada por la sensibilidad, el valor que de a la salida sumado el offset teórico calculado será el valor real que tendría que ver a la salida. Para ello, mediante el potenciómetro R_{G2} vamos a variarlo hasta que para la resistencia puesta a la entrada obtengamos el valor a la salida que indica la siguiente tabla en el valor V_{out} :

	RTD	$m \cdot x$	n	V_{out}
TEORICO V.	98,075	16,7973104	-16,7973104	0
Arbitrario	109,815	18,8080208		2,01071041
RTD real	110,76	18,969871		2,17256061
	99,688	17,073569		0,27625859

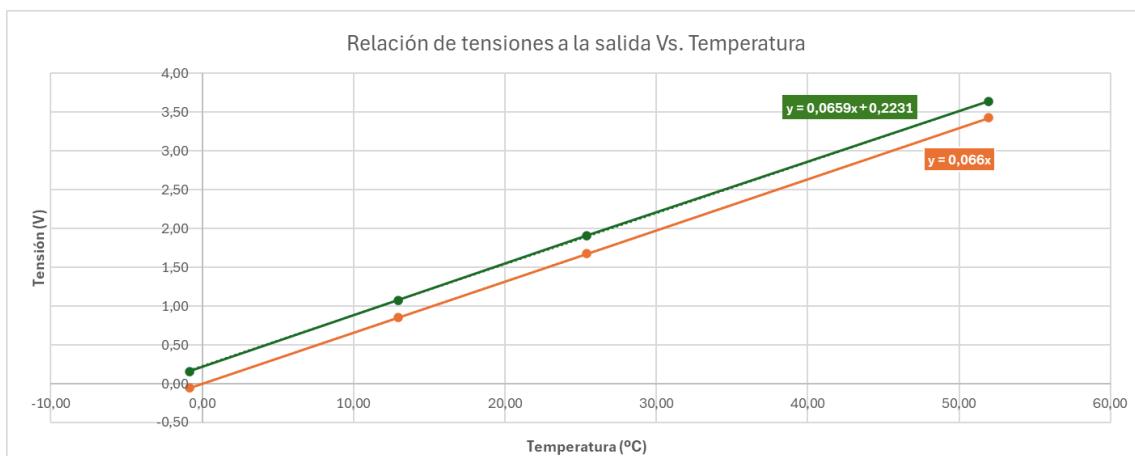
Se comprueba que para la resistencia más pequeña tenemos a la salida un valor de 0,276 V, que corresponde al offset de nuestro circuito. Ahora mismo el acondicionador está ajustado, pero para poder conocer de primera mano las características del acondicionador necesarias para el funcionamiento real de este circuito, vamos a caracterizarlo. Para ello, los estudiantes han escogido cuatro resistencias dentro del intervalo de temperatura que va a trabajar el acondicionador ya que no ha sido posible conseguir más valores resistivos. Para ello, variaremos las RTDs y medimos en la PCBs tanto para V_d como para V_o , obteniendo la siguiente tabla:

R (Ω)	T ($^{\circ}C$)	MEDIDO		CALCULADO		MEDIDO	ERROR NL puente	ERROR NL acondicionador
		V_d medido (mV)	V_o medida (V)	V_d lineal (mV)	V_o lineal (V)			
99,673	-0,85	-7,560	0,160	-1,35	-0,06	-21,18	13,62	0,21
104,98	12,94	21,022	1,077	20,58	0,85	51,22	-30,20	-0,46
109,782	25,41	40,837	1,911	40,42	1,68	46,80	-5,96	-0,09
120	51,95	81,955	3,642	82,64	3,43	44,44	37,52	0,57

Los valores más representativos obtenidos tras esta caracterización son los siguientes:

Av_lineal	41,49	V/V
Av_ajustada	38,8962744	V/V
Incr_Vd	89,515	mV
Incr_Vo	3,482	V
S_Real	0,06594643	V/ $^{\circ}C$
	65,9464259	mV/ $^{\circ}C$

Por lo tanto y para concluir este apartado, la conclusión que obtienen los estudiantes es que el ajuste de sensibilidad es prácticamente el mismo, pero el mayor problema que existe es el **error de no linealidad**, el cuál es de aproximadamente **3 $^{\circ}C$** e imposible de corregir ya que la RTD con la que contamos es de muy mala calidad ya que es de Clase B lo que significa que la precisión deja mucho que desear. La recta que caracteriza dicho acondicionador es la siguiente:



Donde la recta naranja es la ideal y la recta verde es la real, con la ecuación de la recta obtenida tras la caracterización.

- **Análisis del diseño:**

En lo que al circuito analógico para medir la temperatura se refiere, este consistirá en un acondicionador basado en un Amplificador de Instrumentación (AI) junto a un puente de Wheatstone. El objetivo de este es el poder convertir la pequeña variación de valor resistivo que sufre la Pt100 que constituye la RTD, en una variación de tensión y a poder medirla mediante un puerto ADC de la STM32.

En la parte de izquierda del circuito se puede observar cómo se contará con un puente de Wheatstone formado por 4 resistencias: R1, R2, R3 Y la RTD y que se encontrará alimentado con una tensión de 5 voltios positivos que obtendremos de la placa STM32 con el objetivo de que esta sea lo más estable posible.

En lo que a la RTD se refiere, esta consistirá en una Pt100 que contará con una variación resistiva respecto a la temperatura correspondiente con su coeficiente de temperatura, según la ecuación:

$$RTD = 100 \cdot (1 + 0,00385 \cdot T) \Omega$$

Dicha variación resistiva generará una diferencia de tensión entre los nodos de entrada del AI, denominados V_{RTD+} y V_{RTD-}, que es proporcional a la diferencia de temperatura con respecto al equilibrio térmico del puente. Esto será así, ya que el puente se encuentra diseñado de tal modo que cuando el valor resistivo de la RTD es de 100 Ω, correspondiente a 0 °C, este se encontrará en equilibrio, obteniendo así una tensión de 0 V a la salida del acondicionador.

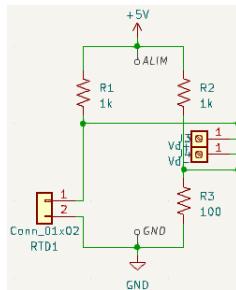


Ilustración: Puente de Wheatstone

En lo que a la etapa de amplificación diferencial se refiere, esta será la encargada de amplificar la pequeña diferencia de tensión obtenida por el puente. Para ello se encontrará compuesta por los operacionales denominados U1 y U2, ambos del tipo OP07. A su vez, estos serán los que conformarán una estructura equivalente a un AI discreto, donde la ganancia será establecida mediante las resistencias externas R4, R5, R6 y R7 bien emparejadas y la resistencia RG1.

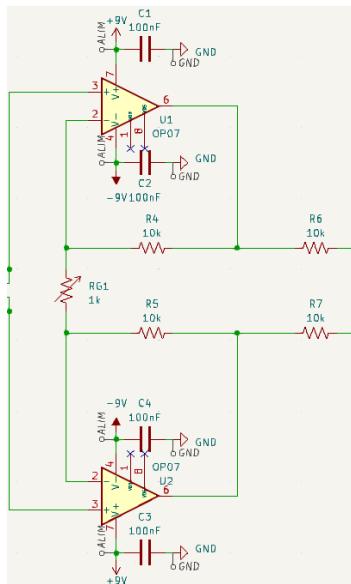


Ilustración: Etapa de amplificación diferencial

En la parte derecha del circuito contaremos con la etapa encargada de generar la tensión de referencia Vref. Dicha etapa se encontrará compuesta por un operacional OP07, denominado U3, que junto con las resistencias R10, R20k1 y la tensión de 5V proporcionarán a su salida una tensión que nos ayudará a ajustar dentro del rango del puerto ADC las tensiones finales.

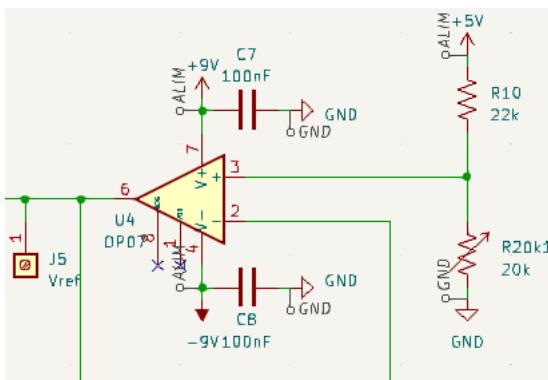


Ilustración: Etapa tensión de referencia

Finalmente, el amplificador U4, el cuál será otro OP07, será el que nos proporcionará la tensión final del circuito, que será la tensión amplificada procedente del puente junto con la tensión de referencia, y que será una tensión que se encontrará dentro del rango del puerto ADC de la placa STM32, entre 0 y 3,3 V.

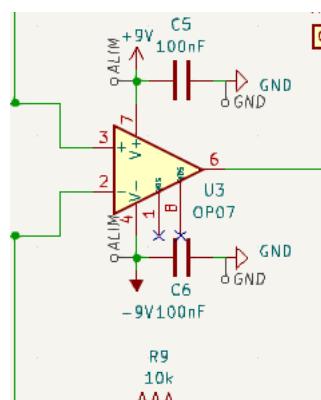
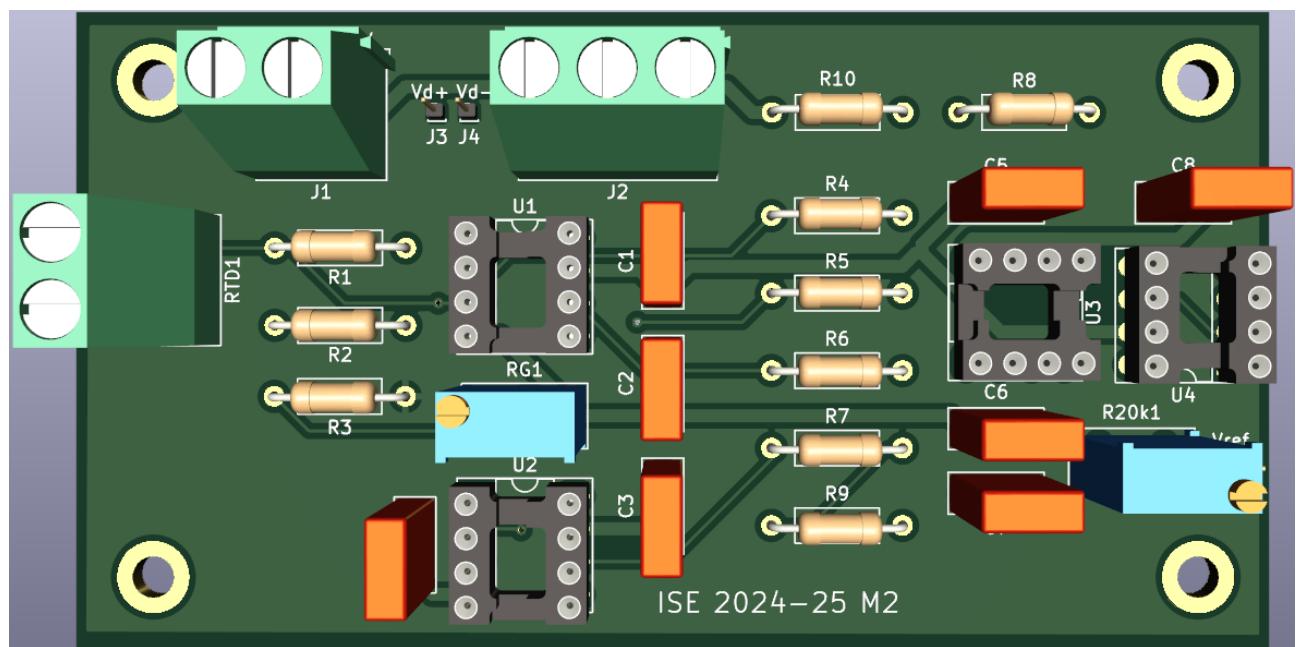
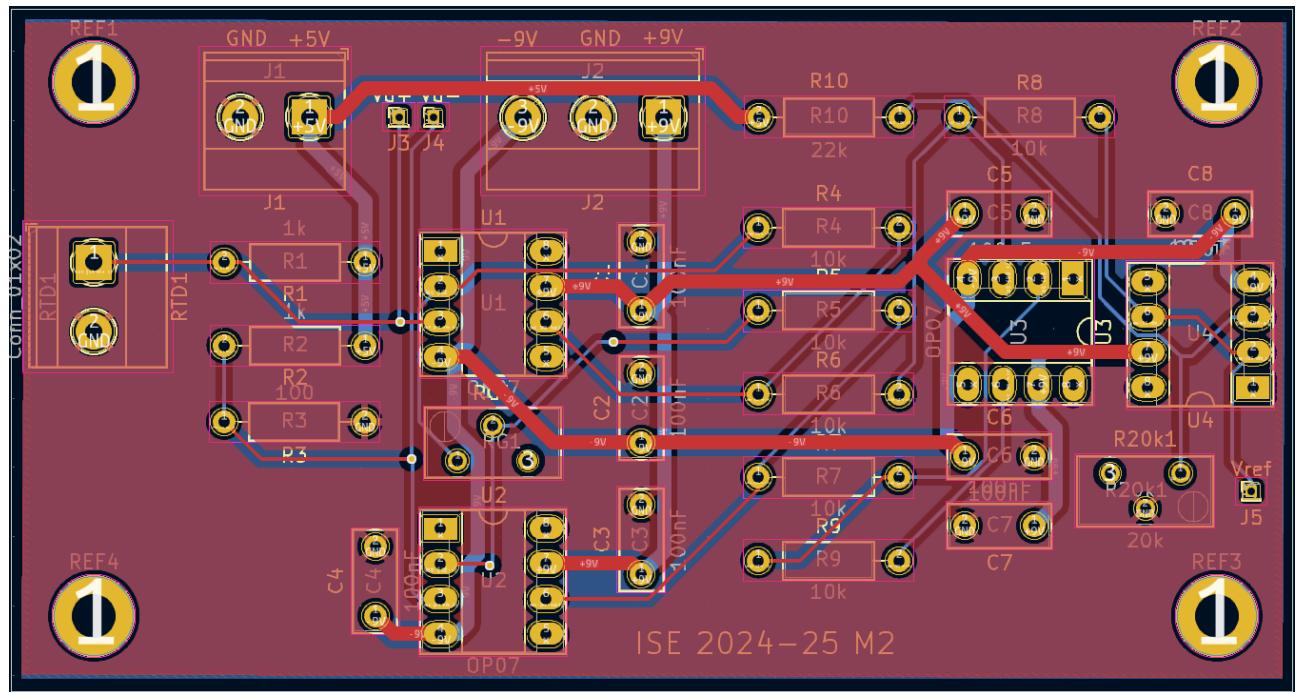
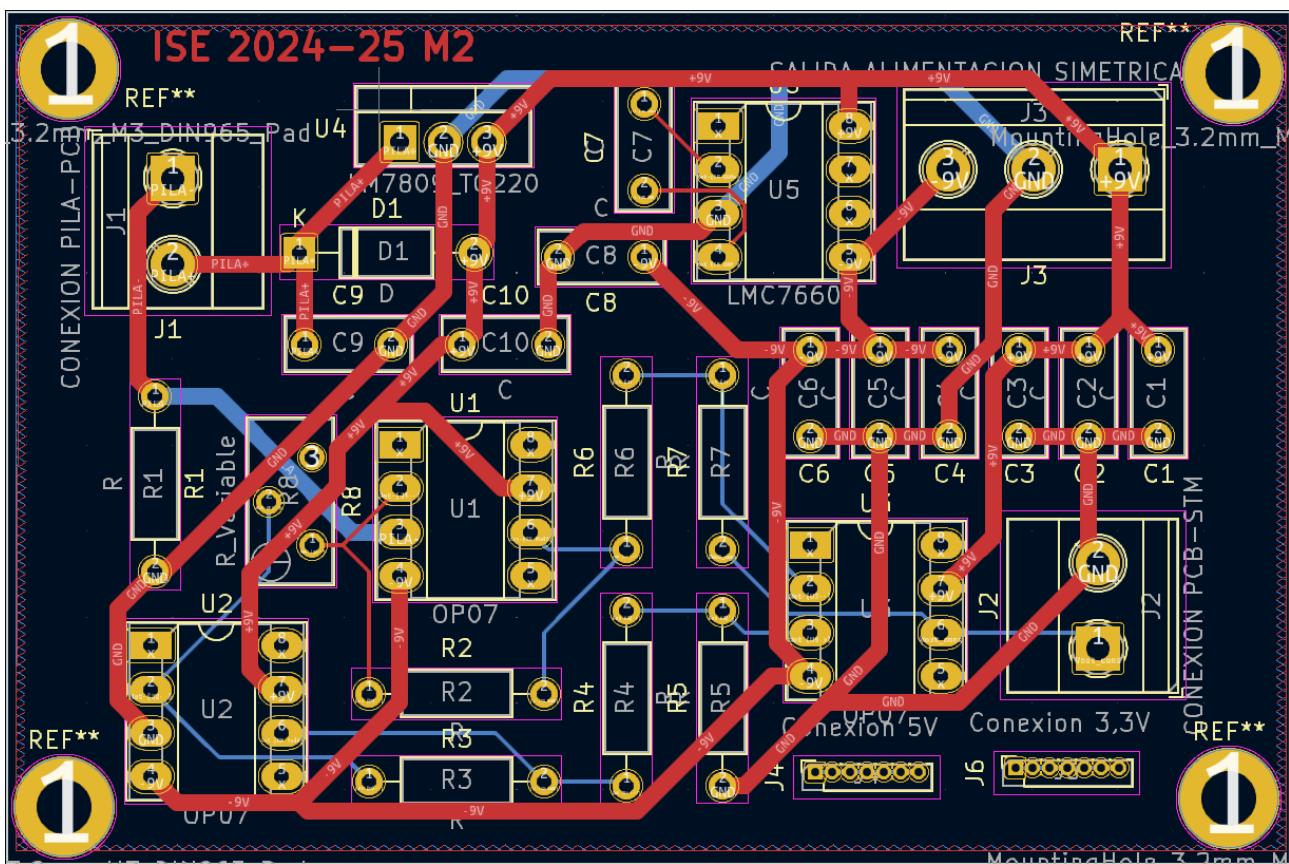
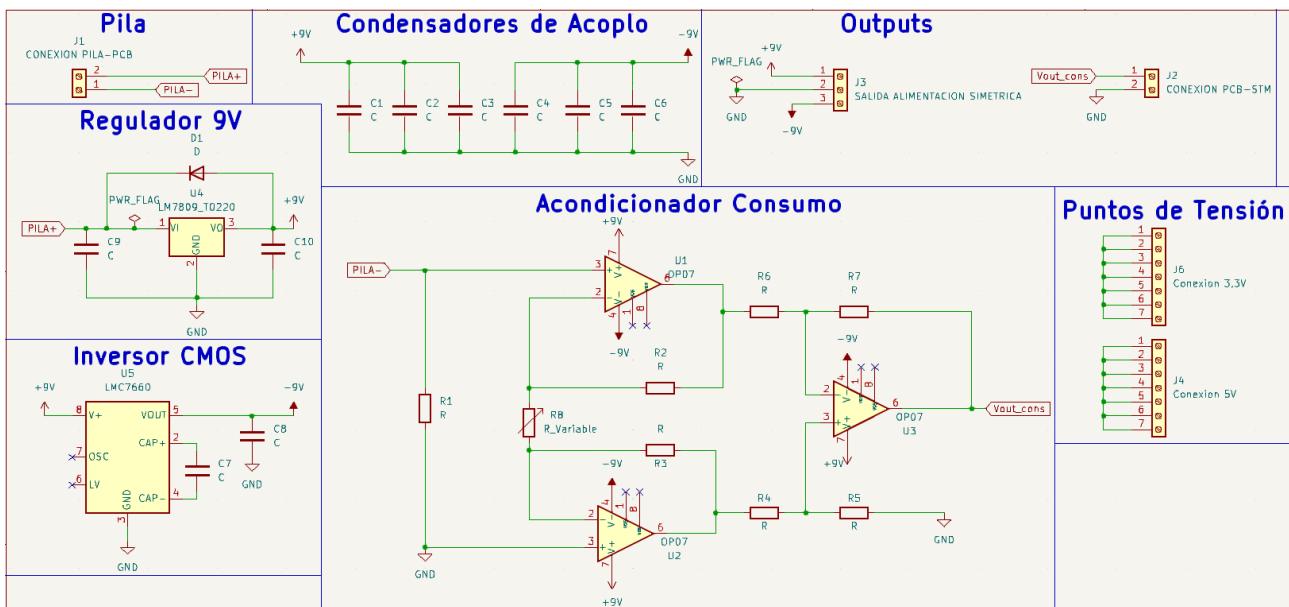


Ilustración Sumador



2.2.2. Acondicionador para el consumo



El propósito de este circuito es medir el consumo eléctrico del sistema completo y de cada uno de sus bloques funcionales. Para ello, se emplea una resistencia shunt de bajo valor ($0,1 \Omega$), a través de la cual se produce una pequeña caída de tensión proporcional a la corriente que circula por el circuito. Dicha caída de tensión, denominada V_d , se mide entre las dos patas de la resistencia shunt. Conociendo su valor, la corriente se calcula directamente mediante la Ley de Ohm:

$$I = \frac{Vd}{R_{shunt}}$$

Para que esta pequeña tensión sea más fácil de medir y registrar, se amplifica utilizando un amplificador instrumental construido con amplificadores operacionales. La salida de este amplificador, denominada V_{out_cons} , representa una versión amplificada de V_d , con una ganancia que debe ser ajustada para cubrir el rango de salida adecuado (de 0 V a 3,2 V). Todas las tierras del sistema están unidas en un punto común conectado a un lado de la resistencia shunt, asegurando una referencia común para todas las mediciones.

Este diseño asegura que la **tensión en modo común (common-mode voltage)** en la entrada del amplificador instrumental sea prácticamente cero. La tensión en modo común se define como:

$$V_{CM} = \frac{(V_+ + V_-)}{2}$$

donde V_+ y V_- son las tensiones aplicadas a las entradas no inversora e inversora del amplificador. En este caso, si una entrada está a 0 V y la otra está, por ejemplo, a 10 mV, entonces $V_{CM}=10\text{ mV}$, un valor muy bajo y completamente dentro del rango de operación seguro del amplificador.

Este detalle no es trivial. La mayoría de los amplificadores operacionales y amplificadores instrumentales tienen **limitaciones estrictas sobre el rango de tensión en modo común permitido**. Si esta tensión se aproxima o supera los límites de alimentación del amplificador, se produce una pérdida de linealidad, distorsión de la señal, e incluso saturación o bloqueo del circuito.

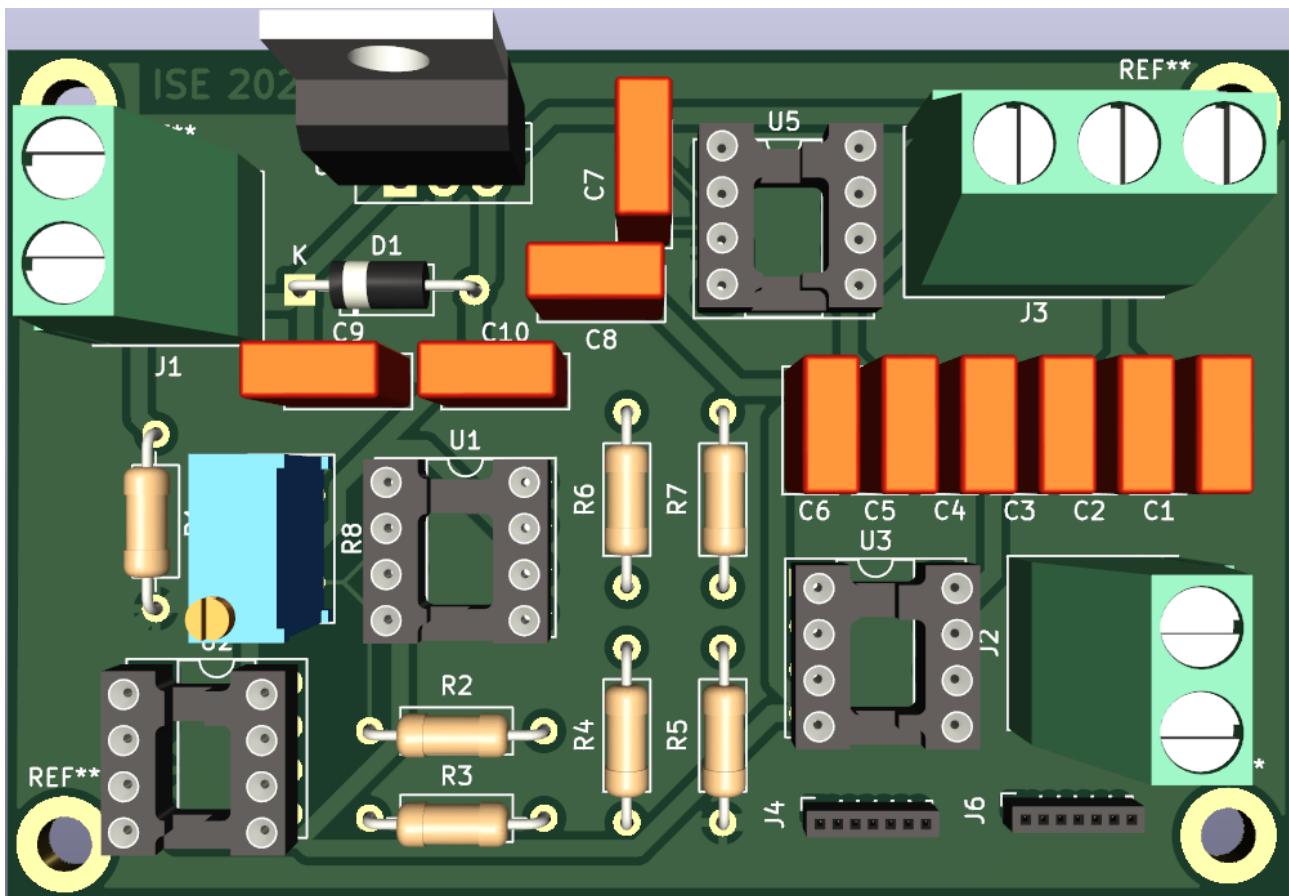
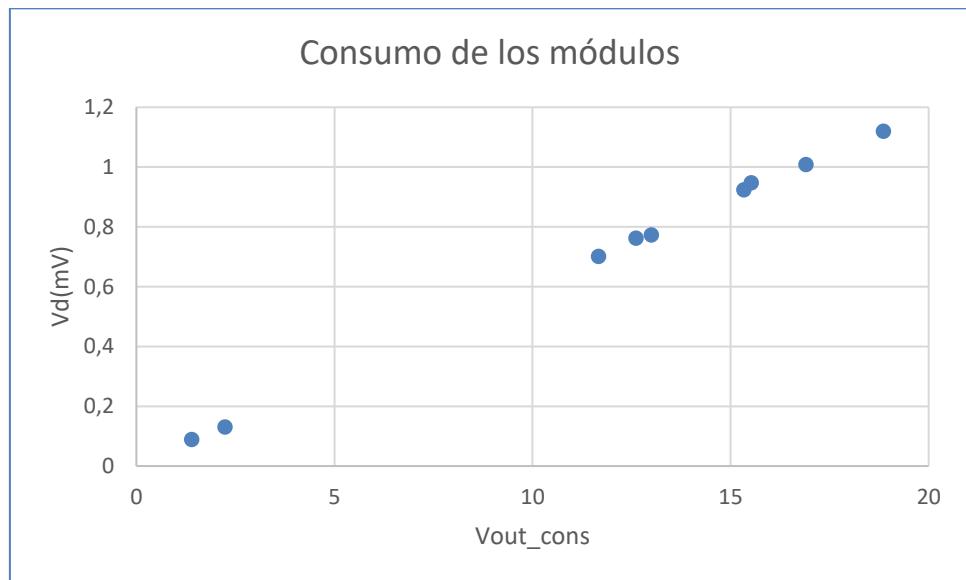
Hemos escogido una posible corriente máxima que circula por la R_{shunt} , de 500 mA, por lo tanto, la ganancia teórica que hemos calculado es la siguiente:

$$G = \frac{V_{outmax} - V_{outmin}}{V_{dmax} - V_{dmin}} = \frac{3,2 - 0\text{ (V)}}{50 - 0\text{ (mV)}} = 64 \left(\frac{V}{V}\right)$$

Hemos ajustado la salida V_{out_cons} para conseguir la ganancia teórica. Una vez ajustada la ganancia hemos medido el consumo de los diferentes bloques que tenemos en el invernadero. Estos datos están recogidos en la siguiente tabla:

MEDIDAS DEL CONSUMO		
MEDIDA CIRCUITO	Vd(mV)	Vout_cons(V)
PCB_CA	1,4	0,0896
PCB_CA + PCB_TEMP	2,244	0,13095
PCB_CA + STM32	11,673	0,70215
PCB_CA + STM32 + BME680	15,527	0,9476
PCB_CA + STM32 + HCR504	13,001	0,77382
PCB_CA + STM32 + BH1750	12,617	0,7628
PCB_CA + STM32 + SENsoRES	16,902	1,009
CIRCUITO COMPLETO	18,865	1,12032
CIRCUITO BAJO CONSUMO	15,34	0,92452

PCB_CA	Placa consumo y alimentación
PCB_TEMP	Placa temperatura
STM32	
BME680	Sensor de humedad y IAQ
HCR504	Sensor ultrasónico
BH1750	Sensor de luminosidad



2.3 Sensores integrados (I2C, SPI, etc).

En este apartado únicamente se explicará cómo funciona cada integrado y un poco de su documentación técnica, sin entrar mucho en detalle. La descripción y funcionamiento de cada integrado se detallará correctamente en el punto [4.2](#) de esta memoria.

2.3.1. Sensor BME680

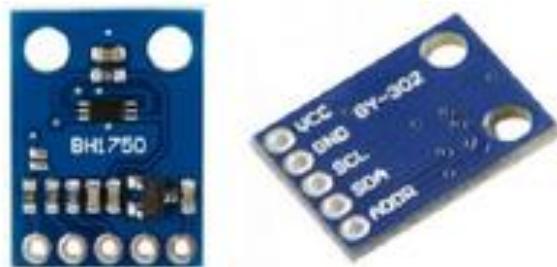
Para poder medir la humedad y calidad del aire del invernadero se ha optado por este sensor, desarrollado por Bosch Sensortec, es un dispositivo muy compacto que permite medir temperatura, humedad, presión además de compuestos orgánicos volátiles. Este puede comunicarse con nuestro microcontrolador a través de dos interfaces, SPI e I2C, pero en este proyecto se va a hacer uso del segundo.



Durante los modos de **WEB** y **PRESENCIAL**, este sensor se encargará de medir constantemente la humedad y la calidad del aire del ambiente. La humedad del sistema se representa en tanto por ciento (%) y la calidad del aire como un entero. Durante el bajo consumo, se dormirá a este sensor llamando a las funciones que lo deshabilitan.

2.3.2. Sensor BH1750

Se trata de un sensor digital de luz ambiental desarrollado por ROHM Semiconductor. Está diseñado para medir la intensidad de luz (luminancia) en lux (lx), con un rango de medición desde 1 lux hasta 65535 lux. Puede tener distintos modos de medición dependiendo de la resolución seleccionada. La interfaz de comunicación de este sensor es **I2C**.



De la misma forma que el sensor anterior, en los dos modos tanto **WEB** como **PRESENCIAL** este sensor estará constantemente midiendo la luminosidad del invernadero. Este valor se podrá ver tanto en el LCD como en la web con una resolución de 0.1 lux.

2.3.3. Sensor HC-SR04

Se trata de un sensor ultrasónico económico y ampliamente utilizado para medir distancias en aplicaciones electrónicas y de robótica. Su funcionamiento se basa en el eco ultrasónico, similar a radares, calculando la distancia al objeto. En este proyecto se va a emplear para medir el nivel de agua de un tanque de agua, ya que es otra aplicación. Todo esto se gestionará mediante un **GPIO** activando y desactivando cuando se desee medir.



Durante el modo **WEB** y **PRESENCIAL**, emitirá un pulso ultrasónico de **40 kHz** a través del pin **TRIG**. Este pulso viaja por el aire y se refleja al chocar con un objeto, en este caso el agua. El sensor recibe el eco mediante el pin **ECHO** y mide el tiempo transcurrido entre la emisión y recepción del pulso, pudiendo medir distancias de 2 a 400 cm. Para hacer mediciones precisas es necesario esperar, al menos 60 ms entre mediciones para evitar interferencias ecos.

2.3.4. Memoria EEPROM AT24C256

A fin de guardar de forma permanente los parámetros más característicos del invernadero se ha escogido la memoria **EEPROM AT24C256**. Este chip es una memoria EEPROM de **256 Kbit**, organizada en 32768 direcciones de 8 bits cada una. Como se ha comentado al principio, su función no es otra que almacenar datos no volátiles, en este caso las 10 últimas medidas realizadas durante el modo **STANDBY** (cuando el usuario no se encuentra en ningún modo). Para ello, emplea el bus de comunicación **I2C**:



Como se ha comentado en la introducción, cada tiempo determinado por los desarrolladores del proyecto se irán almacenando en memoria estos parámetros: HORA, TEMPERATURA, HUMEDAD, LUMINOSIDAD, CALIDAD DEL AIRE y NIVEL DE AGUA. Cada parámetro se guarda en una página

de la memoria, ya que este chip se estructura en **512 páginas de 64 bytes** cada una. Cuando el usuario se encuentre en la página web (únicamente), se leerán las medidas almacenadas, se colocarán en un buffer circular y se mostrará en la ‘subweb’ dedicada a esta implementación.

2.3.5. Lector de tarjetas RFID RC522

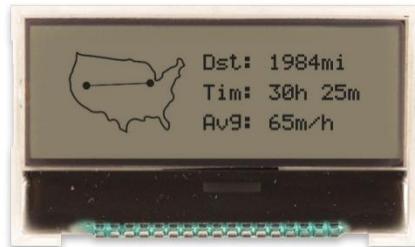
El sensor **RFID RC522** es un módulo lector/escritor **RFID** de bajo costo que opera en la frecuencia de **13,56 MHz**, diseñado para trabajar con etiquetas compatibles con el estándar ISO/IEC 14443A, como las tarjetas MIFARE. Funciona principalmente generando un campo electromagnético que activa tarjetas cercanas, permitiendo así la lectura del identificador único (UID) y otros datos. A continuación, se muestra la imagen del integrado físico:



Está basado en el chip **MFRC522** el cual puede emplear comunicaciones I2C, SPI y UART. Como se mencionó anteriormente, la comunicación empleada en este proyecto es **SPI**. Permite tanto leer como escribir en bloques de memoria de las tarjetas. Es comúnmente usado en sistemas de control de acceso, identificación y automatización. En este proyecto, el RC522 se emplea únicamente para **identificar a usuarios autorizados** cuyo identificador está almacenado en el sistema mediante la lectura de la tarjeta, **activando y permitiendo** el acceso al modo **PRESENCIAL**. En ningún momento se ha planteado el control de acceso con almacenamiento de usuarios. Únicamente se puede acceder a este modo con los identificadores que se encuentran en la tarjeta y llavero.

2.3.6. Display LCD 128x32 px

Este periférico **display LCD 128x32 píxeles** es una pantalla gráfica monocroma de 128 columnas por 32 filas de píxeles, dividido en cuatro páginas, el cual permite mostrar texto, iconos o gráficos simples. Se encuentra incluido en la tarjeta mbed de aplicaciones, el cual incluye dicho display y una **interfaz SPI**, comunicándose con el microcontrolador mediante este mismo protocolo de comunicaciones, permitiendo una transmisión de datos rápida y eficiente:



El uso que tiene este periférico en el proyecto es el de mostrar durante el modo **PRESENCIAL** los parámetros característicos del invernadero en ese mismo momento, además de dar un **feedback** cuando el usuario accede y sale del invernadero tras pasar los **identificadores RFID**. En este modo, en N que se muestran cada **10 segundos** se puede observar lo siguiente: **consumo actual** (primera pantalla); **luminosidad, calidad del aire y nivel de agua** (segunda pantalla); la **temperatura y humedad** (tercera pantalla). Además, en el modo **STANDBY** se muestra una **estimación del consumo** del sistema durante el bajo consumo.

2.3.7. Led RGB

Este periférico es un **LED RGB** el cual funciona mediante la activación a través de **GPIO**. Es necesario aclarar que este led es de **cátodo común** por lo que es necesario activarlo digitalmente con un nivel bajo. Gracias a la combinación de distintos niveles lógicos, podemos obtener un color u otro necesario para la aplicación.



En este proyecto tiene una doble implementación o diferente uso, como se quiera ver. Ambos están dentro del propio modo **PRESENCIAL**. La primera implementación es para indicar que el acceso al invernadero es correcto o erróneo. Si el color mostrado al pasar la tarjeta es **verde** significa que el sistema la ha reconocido y se permite el acceso al invernadero. Si el color mostrado es el **rojo** significa que la tarjeta introducida no ha sido correcta y/o no está incluida en el sistema. La otra implementación es para mostrar el nivel de agua del depósito. Esto simula que el depósito se encuentra lejos del invernadero y el color muestra de forma visual el nivel de agua. Si el color mostrado es **rojo**, significa que el nivel de agua está entre **0% y 20%**; si el color mostrado es **amarillo**, significará que el nivel de agua está entre los valores de **20% y 70%**. En el momento en el que el color mostrado sea el **verde**, significará que el nivel de agua está por encima del **70%**.

2.3.8. Zumbador

Este periférico contiene el funcionamiento más sencillo de todos los sensores integrados. De la misma forma que el anterior, se activa por **GPIO**, ya que es necesario emitir una onda cuadrada (**PWM**) para que funcione. Dicho pitido, que se resume en una frecuencia, depende de cómo se programe digitalmente al propio zumbador. El que está definido en el programa es de **1 kHz**.



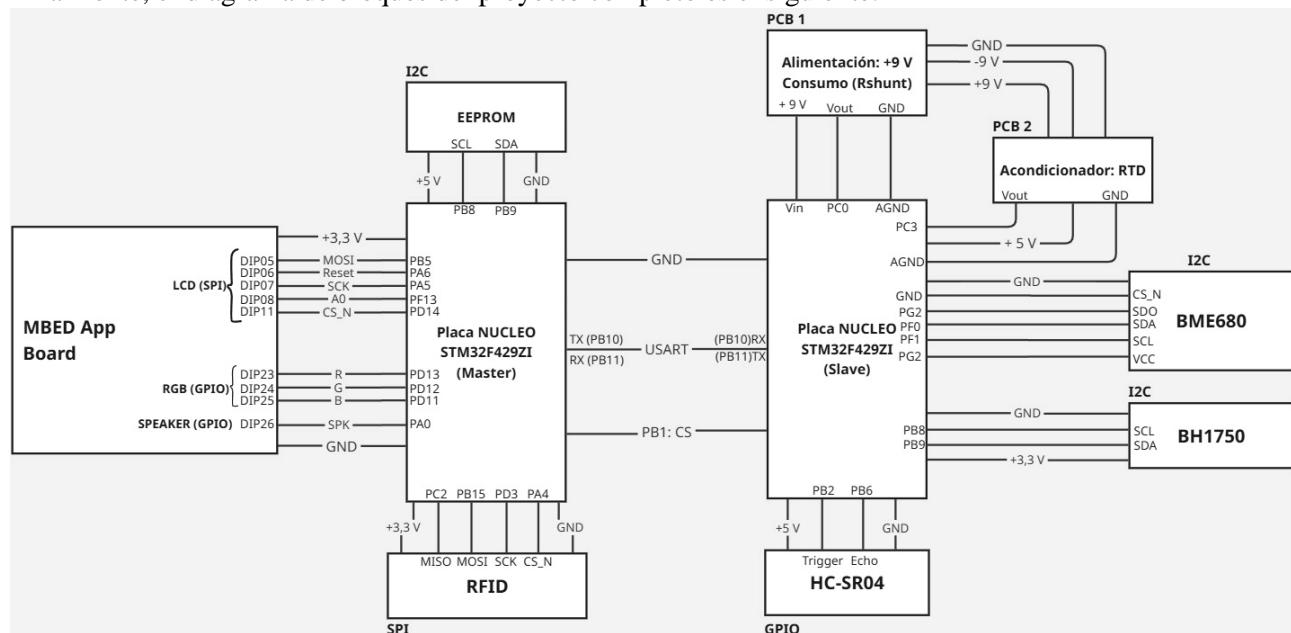
La funcionalidad que tiene en el proyecto es sencilla, ya que solamente se implementa en el modo **PRESENCIAL**. Cuando la identificación del lector RFID es correcta, emite el pitido significando que el usuario puede acceder al invernadero en dicho modo.

3 ESQUEMA ELECTRÓNICO COMPLETO.

Se va a presentar un esquema en Excel de las conexiones de los periféricos a ambas STM's, tanto Máster como Slave:

MASTER										MBED APPLICATION BOARD									
CN8					CN7					Sensor					Tipo pin				
IOREF	1	2	PC8		PC6	1	2	PB8		SPI	GPIO	PA0	DIP26		1				40
	3	4	PC9		PB15	3	4	PB9			MISO	PC2	No aplica		2				39
NRST	5	6	PC10		PB13	5	6	AVDD			MOSI	PB15	No aplica		3				38
IV3	7	8	PC11		PB12	7	8	GND			SCK	PD3	No aplica		4				37
SV	9	10	PC12		PA15	9	10	PA5			CS_N	PA4	No aplica		5				36
GND	11	12	PD2		PC7	11	12	PA6			MOSI	PB5	DIP05		6				35
GND	13	14	PG2		PB5	13	14	PA7			RESET	PA6	DIP06		7				34
VIN	15	16	PG3		PB3	15	16	PD14			SCK	PA5	DIP07		8				33
											A0	PF13	DIP08		9				32
											CS_N	PD14	DIP11		10				31
											R	PD13	DIP23		11				30
											G	PD12	DIP24		12				29
											B	PD11	DIP25		13				28
											VCC	5V	No aplica		14				27
											SCL	PB8	No aplica		15				26
											SDA	PB9	No aplica		16				25
											GND	GND	No aplica		17				24
											UART	TX	PB10	No aplica	18				23
											RX	PB11	No aplica		19				22
															20			21	
																P20		P21	

Finalmente, el diagrama de bloques del proyecto completo es el siguiente:



4 SOFTWARE.

4.1 Interfaz de usuario.

La **interfaz web** desarrollada para el invernadero inteligente tiene como objetivo principal ofrecer al usuario una forma intuitiva y accesible para supervisar en todo momento el estado principal del sistema y consultar información relevante desde cualquier dispositivo con acceso a la red local. El servidor web está alojado en la propia tarjeta núcleo STM32F429ZI, que actúa como servidor web embebido.

Dado a que este proyecto no plantea la interacción entre servidor-usuario, sino simplemente la visualización de los parámetros más característicos del sistema se ha estructurado la web en una **portada principal** y **tres secciones o subpáginas** claramente diferenciadas, accesibles mediante un menú de navegación. Las subpáginas se dividen en: visualización de parámetros en tiempo real, visualización de datos almacenados en la memoria no volátil y, por último, visualización del consumo real.

1. **Página principal:** la única función de esta página es mostrar de una forma visual de que trata la página web. Como se puede comprobar, aparece el nombre del proyecto junto con una barra de navegación para acceder a las distintas subpáginas pinchando sobre el nombre.



2. **Medidas en tiempo real:** esta sección permite al usuario visualizar en directo los valores actuales de los sensores instalados en el invernadero. Entre las variables mostradas se encuentran la temperatura, la humedad ambiental, la calidad del aire, luminosidad y nivel de agua del depósito. Además, contiene tres botones distintos: 'Salir BC', con el que se saca al sistema del bajo consumo haciendo operar a los sensores y enviando la información para que se pueda visualizar en la web; 'Bajo Consumo', haciendo entrar al sistema en bajo consumo y haciendo que los sensores dejen de medir reduciendo así el consumo del sistema; y por último 'Ver Consumo', que cuando se pincha automáticamente se traslada a la subpágina para ver el consumo del sistema.

- 3. Histórico de medidas:** en esta página se muestran los datos almacenados previamente en la memoria EEPROM externa, únicamente cada vez que salta la alarma configurada para cada minuto. Esto permite al usuario realizar un seguimiento de la evolución de las condiciones del invernadero. La información se muestra mediante un buffer circular, en el cual se almacenan las medidas cada vez que el usuario se mete a la web para visualizar las medidas. Como máximo se pueden visualizar 10 medidas en 10 registros temporales distintos. Una vez se llegue a las 10 medidas, la siguiente solapa a la primera medida realizada. Esta subpágina no dispone de botones adicionales.

Histórico de medidas					
H: 21:32:05	T: 26.26°C	H: 44.17 %	WL: 72.67 %	L: 508.33 lx	AQ: 10 PM
H: 21:23:05	T: 25.33°C	H: 44.17 %	WL: 72.67 %	L: 520.00 lx	AQ: 10 PM
H: 21:24:05	T: 25.25°C	H: 44.17 %	WL: 72.53 %	L: 520.83 lx	AQ: 10 PM
H: 21:25:05	T: 25.36°C	H: 44.17 %	WL: 72.53 %	L: 520.83 lx	AQ: 10 PM
H: 21:26:05	T: 25.39°C	H: 44.17 %	WL: 72.53 %	L: 524.16 lx	AQ: 10 PM
H: 21:27:05	T: 25.82°C	H: 44.17 %	WL: 72.67 %	L: 523.33 lx	AQ: 10 PM
H: 21:28:05	T: 26.01°C	H: 44.17 %	WL: 72.53 %	L: 515.83 lx	AQ: 10 PM
H: 21:29:05	T: 25.85°C	H: 44.17 %	WL: 72.67 %	L: 513.33 lx	AQ: 10 PM
H: 21:30:05	T: 26.21°C	H: 44.17 %	WL: 72.67 %	L: 520.83 lx	AQ: 10 PM
H: 21:31:05	T: 26.12°C	H: 44.17 %	WL: 72.53 %	L: 518.33 lx	AQ: 10 PM

Proyecto desarrollado por: Álvaro Salvador, Nizar El Azeouzi, Ignacio Sánchez y Aitor Casado. (UPM)

A la hora de mostrar el histórico, no se ha añadido la fecha a la que se han almacenado dichas medidas ya que se considera que estas medidas se realizan el mismo día en el que se han almacenado.

- 4. Consumo energético del sistema:** esta última sección proporciona información relativa al consumo energético del sistema. Aquí se presentan, en forma de miliamperios, el consumo energético del sistema en tiempo real. Cuando el sistema entra en bajo consumo se muestra una estimación del bajo consumo medida externamente. Cuenta además con dos botones adicionales: 'Ver medidas', que automáticamente en el momento que el usuario pinche se traslada a la subpágina para visualizar las medidas en tiempo real, permitiendo alternar la visualización de estos parámetros; y el botón 'Bajo Consumo', el cual introduce al sistema en bajo consumo mostrando la estimación del bajo consumo.

Consumo Actual del Sistema

Amps	153.100mA
------	-----------

[Bajo Consumo](#)

[Ver Medidas](#)

Proyecto desarrollado por: Álvaro Salvador, Nizar El Azeouzi, Ignacio Sánchez y Aitor Casado. (UPM)

Todas estas subpáginas cuentan con un botón en la esquina superior derecha que permite al usuario volver a la página principal.

4.2 Descripción de cada uno de los módulos del sistema.

En esta sección se van a detallar cada módulo que forma parte de este proyecto, su objetivo, entradas y salidas además de los ficheros de los que forma parte. Por otro lado, se explicará detalladamente cada parte del código con objeto a entender mejor su funcionamiento. Se va a hacer distinción de los módulos que forman parte tanto de la tarjeta maestra como esclava.

1. Tarjeta esclava

- **Módulo nivel de agua:** como se comentó anteriormente, este módulo implementa la medida del nivel de agua de un depósito. Para ello se utiliza el sensor HC-SR04, que funciona mediante GPIO. La descripción funcional es la siguiente:

Nivel del agua	Descripción
Objetivo	Detectar el nivel de agua de un tanque mediante el uso de GPIO.
Entradas	N/A
Salidas	Envía un mensaje por cola con la información del nivel de agua.
Ficheros	hcsr04.c, hcsr04.h

Este módulo emplea los siguientes recursos de RTOS:

Recurso	Valor	Descripción
Hilo	tid_hcsr04	Hilo encargado del sensor ultrasónico HC-SR04
Cola	mid_MsgQueueHCSR04	Cola que envía al módulo PRINC_SLAVE los valores medidos

Este módulo realiza toda la gestión del sensor HC-SR04, el cual se encarga de medir el nivel de agua del tanque del invernadero (que en este caso es un vaso). El modo de funcionamiento de este sensor es mediante GPIO. Dispone de dos pines: ‘Trigger’ (TRIG) y ‘Echo’ (ECH). Por TRIG emite un pulso ultrasónico de 40 kHz, y recoge el rebote de esta onda por el pin ECH, calculando la distancia al objeto midiendo el tiempo que tarda la onda desde que sale hasta que lo recibe. En la función de inicialización del hilo ‘int Init_HCSR04 (void)’ se inicializa tanto el hilo como a la cola de mensajes:

```
int Init_HCSR04 (void) {
    ...
    tid_hcsr04 = osThreadNew(Thread_HCSR04, NULL, NULL);
    if (tid_hcsr04 == NULL) {
        ...
        return(-1);
    }
    ...
    mid_MsgQueueHCSR04 = osMessageQueueNew(MSGQUEUE_HCSR04, sizeof(MSGQUEUE_HCSR04_t), NULL);
    if (mid_MsgQueueHCSR04 == NULL) {
        ...
        return(-1);
    }
    ...
    return(0);
}
```

Posteriormente en la función del hilo ‘void Thread_HCSR04 (void)’ se inicializan a los pines TRIG y ECH para posteriormente realizar la medida y enviarla por la cola de mensajes:

```
void Thread_HCSR04 (void *argument) {
    ...
    GPIO_HCSR04();
    IC_TIM4_Initialization();
    osDelay(100);
    ...
    while (1) {
        // Insert thread code here...
        tx_HCSR04.quantity = getMeasure();
        ...
        osMessageQueuePut(mid_MsgQueueHCSR04, &tx_HCSR04, NULL, 0U);
        ...
        osDelay(1000);
        ...
        //osThreadYield(); ..... // suspend thread
    }
}
```

Las funciones que inicializan los pines son las siguientes: ‘void GPIO_HCSR04 (void)’ y ‘void IC_TIM4 Initialization (void)’:

```
void GPIO_HCSR04 (void) {
    ...
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* Trigger Pulse Initialization */
    /* This pin will go HIGH level for at least 10 us */

    /* PB2 Configuration */
    __HAL_RCC_GPIOB_CLK_ENABLE();

    GPIO_InitStruct.Pin = GPIO_PIN_2;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

    /* Echo Pulse Initialization */
    /* This pin will receive the bounced wavelength */

    /* PB6 Configuration as AF */
    __HAL_RCC_GPIOB_CLK_ENABLE();

    GPIO_InitStruct.Pin = GPIO_PIN_6;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_PULLDOWN;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}
```

```
void IC_TIM4_Initialization (void) {
    ...
    TIM_IC_InitTypeDef sConfigIC;

    /* TIM4 Configuration as IC */
    __HAL_RCC_TIM4_CLK_ENABLE();

    htim4.Instance = TIM4;
    htim4.Init.Prescaler = 83;
    htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim4.Init.Period = 0xFFFF;
    htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;

    HAL_TIM_IC_Init(&htim4);

    sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
    sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
    sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
    sConfigIC.ICFilter = 0x0;

    HAL_TIM_IC_ConfigChannel(&htim4, &sConfigIC, TIM_CHANNEL_1);
    HAL_TIM_IC_Start(&htim4, TIM_CHANNEL_1);
}
```

En la primera se configuran los pines correspondientes. El PB2 corresponde a TRIG y el PB6 corresponde a ECHO. Posteriormente en la segunda función se configura el TIM4 como Input Capture para realizar la medida del pulso de llegada.

La función que realiza el cálculo de la medida del pulso de llegada es ‘float getMeasure(void)’, la cual devuelve un float con el nivel de agua calculado.

```
float getMeasure(void) {
    ...
    uint32_t start = 0, end = 0, timeout = 30000;
    uint32_t counter = 0;
    float width = 0.0f, distance = 0.0f, waterLevel = 0.0f;

    __HAL_TIM_SET_COUNTER(&htim4, 0);
    HAL_TIM_Base_Start(&htim4);

    Init_Sensor(); // Generar pulso

    // Esperar flanco de subida en ECHO
    while (__HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_6) == GPIO_PIN_RESET) {
        counter = __HAL_TIM_GET_COUNTER(&htim4);
        if (counter > timeout) {
            HAL_TIM_Base_Stop(&htim4);
            return -1.0f; // Timeout esperando HIGH
        }
    }
    start = __HAL_TIM_GET_COUNTER(&htim4);

    // Esperar flanco de bajada
    while ((__HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_6) == GPIO_PIN_SET) && (HAL_TIM_GetCounter(&htim4) - start) < timeout) {
        counter = __HAL_TIM_GET_COUNTER(&htim4);
        if (counter - start > timeout) {
            HAL_TIM_Base_Stop(&htim4);
            return -1.0f; // Timeout esperando LOW
        }
    }
    end = __HAL_TIM_GET_COUNTER(&htim4);
    HAL_TIM_Base_Stop(&htim4);

    // Calcular duración
    width = (end >= start) ? (end - start) : (0xFFFF - start + end);
    distance = width / 58.0f;
    waterLevel = ALTURA_DEPO_CM - distance;

    if (waterLevel < 0.0f) return 0.0f;
    else if (waterLevel > ALTURA_DEPO_CM) return 100.0f;
    else return (waterLevel / ALTURA_DEPO_CM) * 100.0f;
}
```

Lo primero que nos encontramos en esta función es que llama a la función ‘void Init_Sensor (void)’ la cual emite un pulso de 10 µs necesarios para que el sensor emita la onda ultrasónica. Dentro de esta función nos encontramos con la función ‘void delay(uint32_t n_microsegundos)’ la cual genera un retraso en microsegundos mediante el TIM7 dependiendo del parámetro de entrada. Posteriormente se espera a que ECHO se ponga a nivel alto (al principio de la recepción) guardando el tiempo en la variable ‘start’. Después espera a que ECHO vuelva a nivel bajo (al final de la recepción) guardando el tiempo en la variable ‘end’ después detiene el temporizador y realiza los siguientes cálculos:

1. Calcula la duración del pulso ECHO:

$$\text{width} = (\text{end} \geq \text{start}) ? (\text{end} - \text{start}) : (0xFFFF - \text{start} + \text{end});$$
2. Convierte dicha duración a una distancia en cm, suponiendo que la onda emitida lleva la velocidad del sonido (343 m/s):

$$\text{distance} = \text{width} / 58.0f;$$
3. Para finalmente obtener la distancia al agua desde el sensor suponiendo que este se encuentra arriba del depósito y lo convierte a porcentaje:

```

    waterLevel = ALTURA_DEPO_CM - distance;

    if (waterLevel < 0.0f) return 0.0f;
    else if (waterLevel > ALTURA_DEPO_CM) return 100.0f;
    else return (waterLevel / ALTURA_DEPO_CM) * 100.0f;

```

En caso de que se desee cambiar el depósito y varíe su tamaño, para que no afecte a la implementación del código y pueda usarse para cualquier depósito dentro del alcance del sensor será necesario modificar la constante:

```
#define ALTURA_DEPO_CM 11.8f
```

Cuyo valor debe introducirse en cm.

- **Módulo sensor de humedad y calidad de aire (BME680):**

Este código implementa un hilo (thread) en un sistema operativo en tiempo real (RTOS) utilizando la biblioteca CMSIS-RTOS2. El hilo se encarga de controlar el flujo de ejecución de un sistema de medida de humedad y calidad de aire (IAQ), para ello se ha utilizado el sensor BME680 que tiene la capacidad de medir temperatura, humedad, presión y gas.

Nivel del agua	Descripción
Objetivo	El objetivo es medir la humedad y la calidad del aire del ambiente
Entradas	N/A
Salidas	Envía un mensaje por cola con la información de la humedad y IAQ
Ficheros	bme680.c, bme680.h

Recurso	Valor	Descripción
Hilo	tid_bme680	Hilo encargado del sensor BME680
Cola	mid_MsgQueueBME680	Cola que envía al módulo PRINC_SLAVE los valores medidos

El hilo se encarga de extraer del sensor BME680, los valores en crudo de temperatura, humedad y gas, los cuales son posteriormente compensados para convertirlos en mediciones precisas y comprensibles. Estos cálculos se han desarrollado siguiendo las indicaciones del datasheet del sensor, donde se describen en detalle tanto las funciones de compensación como la organización de los registros internos. A continuación, se describen los pasos necesarios para un uso adecuado del sensor.

Para establecer la comunicación con el sensor BME680 se ha optado por utilizar el protocolo I²C, debido a su simplicidad de conexión y configuración. Dado que este sensor permite seleccionar entre los modos SPI e I²C mediante el estado del pin CSB, se ha implementado una secuencia de inicialización por software para forzar el modo I²C durante el arranque del sistema.

Concretamente, se ha configurado un pin GPIO como salida digital para controlar el estado del pin CSB del sensor. El procedimiento consiste en llevar el pin a nivel bajo durante un breve periodo (simulando un ciclo de encendido), y posteriormente ponerlo a nivel alto para seleccionar el modo I²C antes de proceder a la inicialización del bus de datos. Este proceso se realiza mediante la siguiente función:

```

150 // INICIALIZACION I2C MASTER -> tid_bme680
151 void Init_I2C_MASTER(void)
152 {
153     osStatus_t status;
154
155     __HAL_RCC_GPIOG_CLK_ENABLE();
156
157     GPIO_InitStruct.Pin = GPIO_PIN_2;
158     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
159     GPIO_InitStruct.Pull = GPIO_NOPULL;
160     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
161     HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
162
163     // Control de alimentación para el sensor
164     HAL_GPIO_WritePin(GPIOG, GPIO_PIN_2, GPIO_PIN_RESET); // CSB = 0 -> SPI (no usar)
165     osDelay(1000); // Simular ciclo de encendido/apagado si se usa GPIO como control de power
166
167     HAL_GPIO_WritePin(GPIOG, GPIO_PIN_2, GPIO_PIN_SET); // CSB = 1 -> MODO I2C
168     osDelay(5);
169
170     status = bme680drv->Initialize(I2C_SignalEvent);
171     status = bme680drv->PowerControl(ARM_POWER_FULL);
172     bme680drv->Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_FAST);
173     bme680drv->Control(ARM_I2C_BUS_CLEAR, NULL);
174     bme680drv->Control(ARM_I2C_OWN_ADDRESS, ADDR);
175     osDelay(5);
176
177     //test_scan_bme680();
178 }

```

Esta configuración garantiza que el sensor BME680 inicie siempre en modo I²C, evitando así posibles conflictos con el protocolo SPI.

Una vez configurado el sensor BME680 en el modo I²C, se han diseñado funciones específicas que permiten realizar las operaciones de escritura y lectura sobre sus registros internos. Estas funciones encapsulan el uso del driver I²C, que nos proporciona CMSIS, donde se encuentran protocolos de escritura y lectura por I²C. A continuación, se describen las funciones implementadas:

```

179 void write_I2C (uint8_t reg, uint8_t value){
180
181     uint8_t cmd[2];
182     cmd[0] = reg;
183     cmd[1] = value;
184
185     bme680drv->MasterTransmit(ADDR, cmd, 2, false);
186     osThreadFlagsWait(S_TRANSFER_DONE, osFlagsWaitAny, osWaitForever);
187
188     //printf("Read reg 0x%02X - valor de escritura: 0x%02X\n", reg, value);
189
190 }
191
192 uint16_t read_I2C (uint8_t reg) {
193     uint8_t measure[2];
194
195     bme680drv->MasterTransmit(ADDR, &reg, 1, true);
196     osThreadFlagsWait(S_TRANSFER_DONE, osFlagsWaitAny, osWaitForever);
197
198     bme680drv->MasterReceive(ADDR, measure, 2, false);
199     osThreadFlagsWait(S_TRANSFER_DONE, osFlagsWaitAny, osWaitForever);
200
201     //printf("Read reg 0x%02X, Measure 0x%02X - 0x%02X\n", reg, measure[0], measure[1]);
202
203     return (measure[0] << 8) | measure[1];
204
205     uint8_t read8 (uint8_t reg) {
206         uint8_t buff;
207
208         bme680drv->MasterTransmit(ADDR, &reg, 1, false);
209         osThreadFlagsWait(S_TRANSFER_DONE, osFlagsWaitAny, osWaitForever);
210         bme680drv->MasterReceive(ADDR, &buff, 1, false);
211         osThreadFlagsWait(S_TRANSFER_DONE, osFlagsWaitAny, osWaitForever);
212
213         //printf("Read reg 0x%02X - valor leido: 0x%02X\n", reg, buff); // agrega esta linea
214
215     return buff;
216 }
217

```

La función *write_I2C* permite escribir un byte en un registro específico del sensor; *read_I2C* lee dos bytes consecutivos desde un registro y los devuelve como una palabra de 16 bits; y *read8* realiza una lectura de un solo byte. Todas las funciones gestionan la transmisión y recepción mediante el driver I²C y sincronizan las operaciones usando flags del sistema operativo en tiempo real para garantizar la correcta finalización de cada transferencia.

Para realizar una medición con el sensor BME680 se debe configurar en modo FORCED. Anteriormente, se debe seguir una secuencia de configuración que asegure condiciones óptimas de precisión y funcionamiento del sensor. El flujo que se debe seguir es el siguiente:

1. Se realiza un reset del sensor asegurando una inicialización conocida antes de que comience la medición. Se escribe el valor 0xB6 sobre el registro de reinicio del sensor *RESET_REG*. Tras la escritura, se espera 5 ms para permitir que el reinicio se complete correctamente.
2. Se selecciona el nivel de sobremuestreo para los tres parámetros ambientales: temperatura, presión y humedad(*osrs_t<2:0>*, *osrs_p<2:0>* y *osrs_h<2:0>*). Para nuestro caso, se ha configurado el sobremuestreo de la humedad x16 y la temperatura con x4, y la presión se ha quedado deshabilitado. Se escribe estos valores sobre los registros *CTRL_HUM* y *CTRL_MEAS*.

3. Se define el coeficiente del filtro IIR que se aplicará a las mediciones de temperatura para reducir el ruido de señal. Esto se configura mediante el campo *filter<2:0>* = 111. Este valor se escribe sobre el registro *CONFIG*.
4. Por último, antes de realizar habilitar la medición se configura el gas. Para ello, se requiere establecer una duración de calentamiento y una temperatura objetivo para la placa calefactora. Esto se realiza de la siguiente manera:
 - Duración de calentamiento: Se define escribiendo en el registro *gas_wait_0<7:0>* el tiempo deseado (por ejemplo, 0x59 para 100 ms).
 - Temperatura de la placa calefactora: Se convierte la temperatura deseada (como 300 °C) en un código y se escribe en *res_heat_0<7:0>*.
 - Selección del perfil de calentamiento: Se configura el registro *nb_conv<3:0>* (por ejemplo, 0x0) para indicar qué conjunto de parámetros usar.
 - Activar medición de gas: Finalmente, se habilita la funcionalidad de medición de gas estableciendo el bit *run_gas* en 1
5. Una vez configurado todos los parámetros, se procede a habilitar la medición estableciendo en el campo *mode<1:0>* en modo **FORCED**. Esta acción inicia una única medición completa, utilizando los parámetros previamente configurados. Una vez finalizada la medición, el sensor vuelve automáticamente al modo de reposo.

```

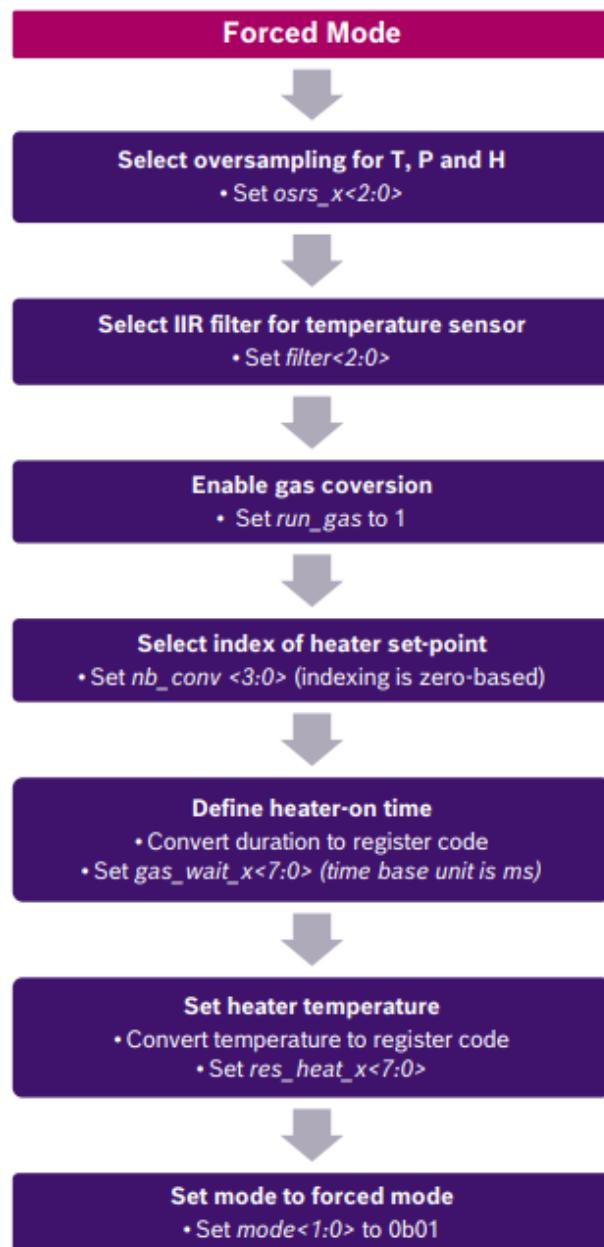
218 void Con_BME680 (void)
219 {
220     // Paso 1: Reset del sensor
221     write_I2C(RESET_REG, 0xE0);
222     osDelay(5); // Tarda 2 ms el reset, poemos un osDelay de 5 ms
223
224     // Paso 2: Leer chip ID
225     uint8_t chip_id = read8(ID_REG);
226
227     // Configurar oversampling de humedad
228     // OVREn = (0b001) para humedad
229     write_I2C(CTRL_HUM, OVRENx16); // HUM oversampling x4 011 = OVREX4
230
231     // Configurar filtro (sin filtro) y standby (0.5ms)
232     // Filtros OFF: 000, T_sb = 000 (0.5 ms) ? valor = 0x00
233     write_I2C(CONFIG, 0x00); // SPI desactivado
234
235     // Configurar oversampling de temperatura y presión + modo
236     // ovrs_t = 100 (x2), ovrs_p = 000 (x1), mode = 00 (forced)
237     uint8_t ctrl_meas = 0x40;
238     write_I2C(CTRL_MEAS, ctrl_meas);
239
240     config_GAS();
241
242     // ovrs_t = 100 (x4), ovrs_p = 000 (x1), mode = 01 (forced)
243     ctrl_meas = 0x20 << 2 | MODE_FORCED;
244     write_I2C(CTRL_MEAS, ctrl_meas);
245
246     //printf("Sensor BME680 configurado para medición ambiental.\n");
247     osDelay(1000);
248 }

```

```

249 void config_GAS(void)
250 {
251     uint8_t res_heat;
252
253     // CONFIGURACIONES DEL TIEMPO DE CALENTAMIENTO DEL SENSOR DE GAS - 100ms
254     write_I2C(GAS_WAIT_0, 0x59);
255
256     // CALENTAMIENTO Y MEDICION DEL SENSOR DE GAS
257     res_heat = calc_res_heat();
258     write_I2C(GAS_WAIT_0, res_heat);
259
260     // HABILITAR LAS MEDIDAS DEL GAS
261     write_I2C(CTRL_GAS_1, RUN_GAS);
262 }

```



El sensor BME680, ya está configurado y listo para realizar una medida. Para obtener la medición de humedad y la calidad del aire (IAQ) con el sensor BME680, se sigue un proceso que combina la lectura de datos crudos del sensor con la aplicación de funciones de compensación utilizando parámetros de calibración internos.

Primero, se leen Los valores en crudo del sensor BME680 se obtienen leyendo directamente los registros internos que almacenan las mediciones sin procesar de temperatura, humedad, presión y gas. Estas lecturas se hacen a través del bus I²C utilizando funciones como `read8()` o `read_I2C()`, que permiten acceder byte a byte a los registros correspondientes. Para formar los datos completos, los bytes leídos se combinan (por ejemplo, desplazando bits y aplicando máscaras) según el formato definido por el fabricante. Estos valores se compensan utilizando algoritmos, proporcionados por el datasheet, que corrigen los efectos de temperatura y calibración para obtener la humedad relativa real en porcentaje.

Luego, se calcula la resistencia del sensor de gas, que refleja la concentración de compuestos volátiles (VOC) en el aire. Esta medida, junto con la humedad, se traduce en una puntuación de calidad del aire (IAQ) que se clasifica en categorías como "Good", "Moderate", "Unhealthy", etc., según un índice basado en límites definidos.

IAQ Index	Air Quality	Impact (long-term exposure)	Suggested action
0 – 50	Excellent	Pure air; best for well-being	No measures needed
51 – 100	Good	No irritation or impact on well-being	No measures needed
101 – 150	Lightly polluted	Reduction of well-being possible	Ventilation suggested
151 – 200	Moderately polluted	More significant irritation possible	Increase ventilation with clean air
201 – 250	Heavily polluted	Exposition might lead to effects like headache depending on type of VOCs	optimize ventilation
251 – 350	Severely polluted	More severe health issue possible if harmful VOC present	Contamination should be identified if level is reached even w/o presence of people; maximize ventilation & reduce attendance
> 351	Extremely polluted	Headaches, additional neurotoxic effects possible	Contamination needs to be identified; avoid presence in room and maximize ventilation

- Sensor de luminosidad (BH1750):

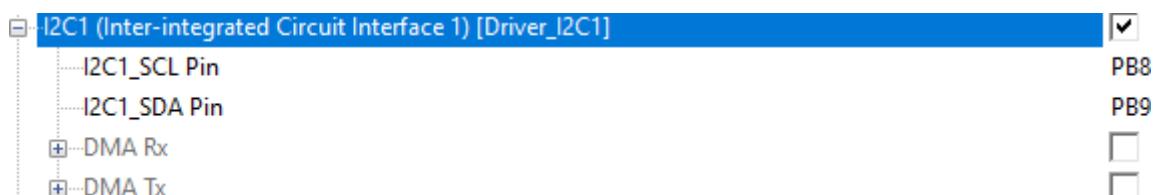
Este módulo implementa la gestión completa del sensor de luminosidad BH1750, cuyo objetivo es permitir la lectura periódica de los niveles de luz en el invernadero a través de una interfaz I²C. La funcionalidad incluye la configuración del bus I²C, la inicialización del sensor y la adquisición recurrente de datos de luminosidad, los cuales se almacenan y envían mediante una cola de mensajes (RTOS) para su posterior uso por otros módulos del sistema tal y como se muestra a continuación:

Luminosidad	Descripción
Objetivo	Medir la luminosidad del ambiente
Entradas	N/A
Salidas	Envía por cola el mensaje con el valor de la luminosidad
Ficheros	bh1750.c, bh1750.h

En lo que, a los recursos del sistema operativo, este hilo hará uso de un hilo para inicializar y obtener medidas periódicas del sensor y una cola para enviar los valores medidos al hilo principal del SLAVE, tal y como se muestra a continuación:

Recurso	Valor	Descripción
Hilo	tid_ThBH1750	Hilo encargado del sensor BH1750
Cola	mid_MsgQueueBH1750	Cola que envía al módulo PRINC_SLAVE los valores medidos

Tal y como mencionamos anteriormente, este módulo será el que lleve a cabo toda la gestión del sensor BH1750, que será el encargado de medir la luminosidad del invernadero. El modo de funcionamiento de este sensor es mediante el uso del protocolo I²C. Para ello, emplearemos el pin PB8 como pin SCL y el pin PB9 como pin SDA de la comunicación, tal y como se muestra a continuación.



La función `'int Init_ThBH1750 (void)'` será aquella en la que tendrá lugar tanto la creación del hilo que se encargará de toda la gestión del sensor, como la cola de mensajes por la que enviaremos los valores medidos.

```

int Init_ThBH1750 (void) {
    tid_ThBH1750 = osThreadNew(ThBH1750, NULL, NULL);
    if (tid_ThBH1750 == NULL) {
        return(-1);
    }

    mid_MsgQueueBH1750 = osMessageQueueNew(MSGQUEUE_BH1750, sizeof(MSGQUEUE_BH1750_t), NULL);
    if (mid_MsgQueueBH1750 == NULL) {
        return(-1);
    }

    return(0);
}

```

La función principal del módulo será la denominada 'void ThBH1750 (void *argument)'. En dicha función será donde se gestionará toda la funcionalidad del módulo. En primer lugar, inicializará el bus I²C, tras esto, inicializará y configurará el sensor de luminosidad y finalmente, dentro del bucle while(1) llamaremos cada segundo a la función 'Brightness_Reading()' con el objetivo de poder conocer así la medida de luminosidad realizada por el sensor de manera periódica y enviarla por la cola de mensajes correspondiente.

```

void ThBH1750 (void *argument) {
    Init_I2C();
    Init_BH1750();

    tx_BH1750.lum = 0.0;

    while(1){
        // Insert thread code here...

        tx_BH1750.lum = Brightness_Reading();
        osMessageQueuePut(mid_MsgQueueBH1750, &tx_BH1750, NULL, 0U);

        osDelay(1000);
        //osThreadYield(); // suspend thread
    }
}

```

La función encargada de la correcta inicialización y configuración del protocolo de comunicación I²C, será la denominada 'void Init_I2C (void)'. En dicha función será donde inicialicemos el controlador I²C, activaremos la alimentación del periférico, configuraremos la velocidad del bus en el modo rápido (400 kHz) y limpiaremos el bus I²C con el fin de prevenir bloqueos.

```

/* Initialize I2C */
void Init_I2C (void) {
    /* Initialize I2C peripheral */
    I2Cdrv->Initialize (I2C_SignalEvent); // Inicializa el controlador I2C

    /* Power-on I2C peripheral */
    I2Cdrv->PowerControl (ARM_POWER_FULL); // Activa la alimentación del I2C

    /* Configure I2C bus */
    I2Cdrv->Control      (ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_FAST); // Configura velocidad rápida (400 kHz)
    I2Cdrv->Control      (ARM_I2C_BUS_CLEAR, 0); // Limpia el bus I2C si está bloqueado
}

```

La función 'void Init_BH1750(void)' será la encargada de realizar la correcta inicialización y configuración del sensor con el fin de dejarlo listo para realizar medidas.

```

/* Inicializacion del sensor BH1750*/
void Init_BH1750(void){

    // Comando para encender el sensor
    uint8_t encender_sensor = 0;

    encender_sensor = BH1750_POWER_ON;

    I2Cdrv->MasterTransmit(BH1750_ADDR, &encender_sensor, 1, false);
    osThreadFlagsWait(TRANSF_DONE_BH1750, osFlagsWaitAny,osWaitForever);

    // Comando para establecer la resolucion
    uint8_t data_transmit = 0;

    data_transmit = CONTINUOSLY_H_RESOLUTION;

    I2Cdrv->MasterTransmit(BH1750_ADDR, &data_transmit, 1, true);
    osThreadFlagsWait(TRANSF_DONE_BH1750, osFlagsWaitAny,osWaitForever);

}

```

En dicha función, como primera interacción con el sensor, enviaremos el comando correspondiente para encender al mismo. Dicho comando será el 0x01, y este lo enviaremos al registro 0x23 del sensor, que corresponde con el registro ADDR del mismo cuando este cuenta con una tensión menor a 0.3*Vcc por dicho pin. Tras esto, le enviaremos el comando correspondiente a la configuración con la que desearemos que este cuente para realizar las medidas. En nuestro caso el comando enviado será el 0x10, el cuál configurará al sensor de tal forma que este realice medidas de manera continua con una resolución de 1 lx, tal y como se muestra a continuación:

H-Resolution Mode Resolution	rHR	-	1	-	lx	
------------------------------	-----	---	---	---	----	--

Finalmente, la función denominada '*float Brightness_Reading (void)*' será aquella que nos devolverá el valor de luminosidad medido por el sensor e el momento que se lo solicitemos.

```

/* Funcion para leer la medida de luminosidad*/
float Brightness_Reading (void){

    uint16_t lux = 0;
    uint8_t brightness[2] = {0};
    float temp = 0.0;

    I2Cdrv->MasterReceive(BH1750_ADDR, brightness, 2, false);
    osThreadFlagsWait(TRANSF_DONE_BH1750, osFlagsWaitAny,osWaitForever);

    // Validar datos antes de convertir
    if (brightness[0] == 0xFF && brightness[1] == 0xFF) {
        return 0;
    }

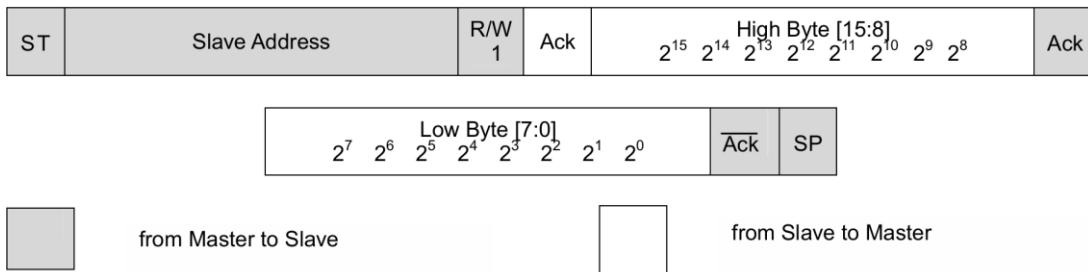
    lux = ((brightness[0] << 8) | brightness[1]); // Convertir a 16 bits
    temp = (float) lux / 1.2;

    return temp; // Conversión a lux según el datasheet
}

```

En dicha función enviaremos el comando de lectura al sensor indicando su dirección ADDR, y posteriormente indicamos donde queremos almacenar los valores recibidos del mismo. A su vez, realizaremos una comprobación para ver si el valor devuelto por el sensor es un valor válido de luminosidad. Continuando, debido a que el sensor nos devolverá la medida de luminosidad en formato Big Endian, por lo que concatenaremos los bits más significativos situados en la posición 0 con los menos significativos y lo igualaremos a una variable. Finalmente, el valor de luminosidad leído una vez realizada la concatenación, será necesario dividirlo entre 1.2 por especificación del fabricante, para obtener así el valor real de luminosidad.

4) Read Format



ex)

High Byte = "1000_0011"
 Low Byte = "1001_0000"
 $(2^{15} + 2^9 + 2^8 + 2^7 + 2^4) / 1.2 \approx 28067 [\text{lx}]$

- Convertidores analógicos-digitales (ADCs consumo y temperatura):

Este módulo implementa los dos ADC's (Analog-Digital Converter) usados en este proyecto. Cada uno convierte el voltaje analógico a voltaje digital respectivos al consumo y a la temperatura que se obtienen de las dos PCB's diseñadas por los estudiantes. La descripción funcional del módulo es la siguiente:

Temperatura y Descripción Consumo	
Objetivo	Obtiene la tensión de entrada de los acondicionadores de temperatura y consumo
Entradas	Tensión que proviene de ambos acondicionadores
Salidas	Envía por cola los mensajes de temperatura y consumo, ambos en float .
Ficheros	adcs.c, adcs.h

Y los recursos del sistema operativo que utiliza este módulo son:

Recurso	Valor	Descripción
Hilo	tid_ThADCs	Hilo encargado de los puertos ADCs
Cola	mid_MsgQueueADCs	Cola que envía al módulo PRINC_SLAVE los valores medidos

Este módulo realiza toda la gestión de los ADC's que incluye la tarjeta STM32F429ZI. En concreto, se usa únicamente el ADC1 para las dos entradas analógicas usando dos canales distintos. En la función del hilo 'int Init_ThADCs (void)' se inicializa tanto al hilo como a la cola de mensajes descrita arriba. De la tarjeta núcleo, se usan los pines PC0 para el consumo y el PC3 para la temperatura. Estos se inicializan en las funciones 'void RTD_pin_F429ZI_config ()' y 'void CONSUMPTION_pin_F429ZI_config ()':

```

void RTD_pin_F429ZI_config(){
→ · GPIO_InitTypeDef GPIO_InitStruct = {0}; void CONSUMPTION_pin_F429ZI_config(){
→ → · HAL_RCC_ADC1_CLK_ENABLE();
→ → · HAL_RCC_GPIOC_CLK_ENABLE();
→ →
→ /* · PC3 -----> · ADC1_IN13 */
→ →
· GPIO_InitStruct.Pin = GPIO_PIN_3;
· GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
· GPIO_InitStruct.Pull = GPIO_NOPULL;
· HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
→
}
→ · GPIO_InitTypeDef GPIO_InitStruct = {0};
→ → · HAL_RCC_ADC1_CLK_ENABLE();
→ → · HAL_RCC_GPIOC_CLK_ENABLE();
→ →
→ /* · PC0 -----> · ADC1_IN10 */
→ →
· GPIO_InitStruct.Pin = GPIO_PIN_0;
· GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
· GPIO_InitStruct.Pull = GPIO_NOPULL;
· HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
→
}

```

Posteriormente en la función del hilo ‘`void ThADCs (void *argument)`’ es donde se realiza toda la implementación del módulo. Primeramente, se inicializan a ambos pines del ADC, para después llamar a la función ‘`int ADCs_Init_Single_Conversion(ADC_HandleTypeDef *hadc, ADC_TypeDef *ADC_Instance)`’ donde se configura el ADC1 para realizar la conversión deseada:

```
|void ThADCs (void *argument) {
→
→RTP_pin_F429ZI_config();
→CONSUMPTION_pin_F429ZI_config();
→ADCs_Init_Single_Conversion(&adchandle_adcs, ADC1); //ADC1 configuration
→
· tx_ADCs.RTD_Vol = 0;
· tx_ADCs.CONSUMPTION_Vol = 0;
·
|· while (1) {
··· //Insert thread code here...
··· tx_ADCs.RTD_Vol =(uint16_t) ADCs_getVoltage(&adchandle_adcs, ·13); //get values from channel 13->ADC123_IN13
→
→tx_ADCs.CONSUMPTION_Vol = (uint16_t)ADCs_getVoltage(&adchandle_adcs, ·10); //get values from channel 13->ADC123_IN13
···
··· osMessageQueuePut(mid_MsgQueueADCs, &tx_ADCs, ·NULL, ·OU);
→
→osDelay(1000);
→//osThreadYield(); ..... //suspend thread
· }
}
```

Dentro del bucle while se asignan a las variables de la cola a las funciones que devuelven las respectivas tensiones de cada pin. La función ‘`uint32_t ADCs_getVoltage(ADC_HandleTypeDef *hadc, uint32_t Channel)`’ devuelve en 32 bits la tensión obtenida de cada pin asociado a un canal distinto dentro del mismo ADC. El PC3 (temperatura) se asocia al canal 13 y el PC0 (consumo) se asocia al canal 10. Finalmente lo envía por la respectiva cola al módulo principal del slave.

- Módulo comunicación Slave:

El módulo implementa la comunicación UART entre la placa B (configurada como Slave) y la placa A (máster). El sistema está diseñado para que la placa A envíe comandos estructurados de 3 bytes, los cuales son procesados por la placa B. Esta verifica la integridad de los datos (evitando corrupción) y genera una respuesta de 14 bytes que puede incluir tanto lecturas de sensores como comandos especiales como SLEEP o WAKE UP.

Comunicación Slave	Descripción
Objetivo	Comunicación UART entre la placa B y la placa A
Entradas	Trama de 3 bytes proveniente desde la placa A
Salidas	Trama de respuesta de 14 bytes desde la placa B
Ficheros	com.c, com.h

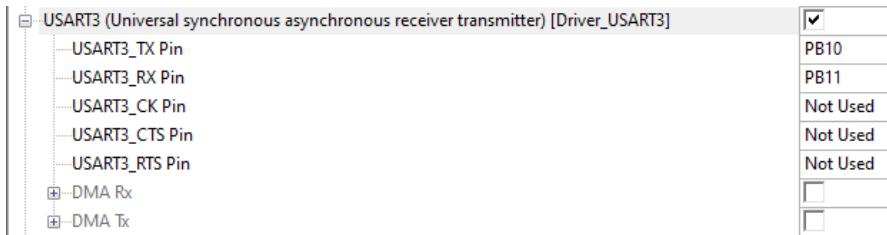
Entradas UART:

- Se recibe una trama de 3 bytes desde la Placa A (Máster) con la siguiente estructura:
 - Byte 1: Inicio de trama (0x7E)
 - Byte 2: Código de comando (0x01, 0xAA, 0xBB)
 - Byte 3: Fin de trama (0x7F)

Salidas UART:

- Se envía una trama de respuesta de 14 bytes desde la Placa B (Slave), estructurada según el comando recibido. La trama contiene:
 - Datos codificados de sensores (temperatura, humedad, luminosidad, calidad del aire, nivel de agua, consumo)
 - Código de control (comando de error)
 - Byte de inicio y fin de trama

El modo de funcionamiento de esta comunicación es mediante el uso del protocolo UART. Para ello, emplearemos el pin PB10 como línea de transmisión (TX) y el pin PB11 como línea de recepción (RX) de la comunicación, tal y como se muestra a continuación.



Recurso	Valor	Descripción
Hilo	TID_UartTx	Hilo encargado de enviar respuestas
Hilo	TID_UartRx	Hilo encargado de recibir y validar datos
Cola	mid_Slave_Com_MsgQueue	Cola que recibe los datos de las medidas de todos los sensores y ADC's

Para garantizar el funcionamiento correcto y eficiente, la implementación utiliza dos hilos independientes:

- **UartRx:** Escucha constantemente por comandos UART. Valida la trama recibida (inicio y fin correctos). Si la trama es válida, notifica al hilo UartTx para que procese y responda; si es incorrecta, notifica igualmente, pero marca error.
- **UartTx:** Espera los datos de sensores desde la cola de mensajes y responde cuando UartRx le notifica que un comando ha llegado. Codifica los datos de sensores y envía la respuesta UART.

La sincronización entre ambos hilos se realiza mediante señales (osThreadFlags) e interrupciones (USART_Callback), lo que asegura una comunicación fluida y sin bloqueos.

Mecanismo de Colas:

- La cola de entrada mid_Slave_Com_MsgQueue recibe los datos de todos los sensores en una estructura MSGQUEUE_OBJ_MEAS.
- Los datos de esta cola son utilizados por el hilo UartTx para generar las tramas de respuesta correspondientes.

La función int Init_ThCom (void):

```

int Init_ThCom (void) {
    /* Hilo encargado de enviar info al PC */
    TID_UartTx = osThreadNew(UartTx, NULL, NULL); // Hilo de transmisión
    if (TID_UartTx == NULL) {
        return(-1);
    }

    /* Hilo encargado de recibir info del PC */
    TID_UartRx = osThreadNew(UartRx, NULL, NULL); // Hilo de recepción
    if (TID_UartRx == NULL) {
        return(-1);
    }

    UART_Init(); // Inicializa el periférico UART

    return(0);
}

```

Esta función se encarga de la inicialización del módulo de comunicación UART. Dentro de ella se lleva a cabo la creación de los dos hilos encargados de la transmisión y recepción de datos entre la Placa B (slave) y la Placa A (master), así como la inicialización del periférico UART.

En primer lugar, se crea el hilo de transmisión (UartTx) y el hilo de recepción (UartRx). Tras ello, se llama a la función UART_Init() con el objetivo de inicializar y configurar el periférico UART.

La función void UART_Init (void):

```
67 void UART_Init(void) {
68     // 1. Inicialización
69     int32_t status;
70
71     status = UARTdrv->Initialize(USART_Callback);
72     if(status != ARM_DRIVER_OK) {
73         // printf("[UART] Error en Initialize: %d\n", status);
74         return;
75     }
76     // 2. Encender el periférico
77     status = UARTdrv->PowerControl(ARM_POWER_FULL);
78     if(status != ARM_DRIVER_OK) {
79         // printf("[UART] Error en PowerControl: %d\n", status);
80         return;
81     }
82
83
84     // 3. Configurar parámetros UART
85     status = UARTdrv->Control(
86         ARM_USART_MODE_ASYNCHRONOUS | // Modo UART (asíncrono)
87         ARM_USART_DATA_BITS_8 | // 8 bits de datos
88         ARM_USART_PARITY_NONE | // Sin paridad
89         ARM_USART_STOP_BITS_1 | // 1 bit de parada
90         ARM_USART_FLOW_CONTROL_NONE, // Sin control de flujo
91         9600 // Baudrate
92     );
93
94     if(status == ARM_DRIVER_OK) {
95         // printf("[UART] Inicializado correctamente a 115200 baudios\n");
96         uart_initialized = true;
97     } else {
98         // printf("[UART] Error en Control: %d\n", status);
99     }
100
101    // 4. Habilitar TX y RX
102    status = UARTdrv->Control(ARM_USART_CONTROL_TX, 1);
103    if (status != ARM_DRIVER_OK) {
104        // printf("[UART] Error al habilitar TX: %d\n", status);
105        return;
106    }
107    // Habilitar interrupción por recepción usando el callback (no IRQ)
108    status = UARTdrv->Control(ARM_USART_CONTROL_RX, 1);
109    if (status != ARM_DRIVER_OK) {
110        // printf("[UART] Error al habilitar RX: %d\n", status);
111        return;
112    }
113 }
```

Esta función configura el módulo UART para su operación en modo asíncrono. El proceso comienza inicializando el controlador mediante `UARTdrv->Initialize`, vinculándolo a la función de interrupción `USART_Callback` para gestionar eventos de transmisión y recepción. A continuación, se activa el periférico con `PowerControl(ARM_POWER_FULL)` para suministrar la alimentación necesaria.

La configuración específica del UART incluye: modo asíncrono, formato de 8 bits de datos sin paridad, 1 bit de parada, sin control de flujo y velocidad fija de 9600 baudios. Una vez aplicados estos parámetros, se activan las líneas de transmisión (`ARM_USART_CONTROL_TX`) y recepción (`ARM_USART_CONTROL_RX`), dejando el periférico listo para comunicación bidireccional.

La función int32_t UART_ListenCommand(uint8_t* command_buffer):

```
118 int32_t UART_ListenCommand(uint8_t* command_buffer) {
119     if (!uart_initialized) {
120         // printf("[UART] Error: No inicializado antes de recibir comando\n");
121         return ARM_DRIVER_ERROR;
122     }
123     // printf("[UART] Esperando comando...\n");
124
125     int32_t result = UARTdrv->Receive(command_buffer, 3);
126     if (result != ARM_DRIVER_OK) {
127         // printf("[UART] Error al iniciar recepción del comando: %d\n", result);
128         return result;
129     }
130
131     osThreadFlagsWait(ARM_USART_EVENT_RECEIVE_COMPLETE, osFlagsWaitAny, osWaitForever);
132
133
134     return ARM_DRIVER_OK;
135 }
```

Esta función tiene como objetivo iniciar la recepción de un comando desde la Placa A (máster) y almacenarlo en el buffer `command_buffer`, que debe tener capacidad para 3 bytes.

Primero, la función verifica si el periférico UART ha sido inicializado correctamente. En caso contrario, devuelve un error.

Si todo es correcto, se llama a UARTdrv->Receive() para comenzar la recepción de los 3 bytes que forman el comando. Una vez que se ha completado la recepción, se espera la correspondiente señal de evento (mediante osThreadFlagsWait) que será enviada desde el callback USART_Callback cuando se detecte ARM_USART_EVENT_RECEIVE_COMPLETE.

La función int32_t UART_SendData (uint8_t* datatx):

```

137 int32_t UART_SendData(uint8_t* datatx) {
138     if (!uart_initialized) {
139         // printf("[UART] Error: No inicializado antes de enviar temperatura\n");
140         return ARM_DRIVER_ERROR;
141     }
142
143     int32_t result = UARTdrv->Send(datatx, 14);
144     if (result != ARM_DRIVER_OK) {
145         // printf("[UART] Error al enviar temperatura: %d\n", result);
146         return result;
147     }
148
149     osThreadFlagsWait(ARM_USART_EVENT_SEND_COMPLETE, osFlagsWaitAny, osWaitForever);
150
151     return ARM_DRIVER_OK;
152 }
153

```

Esta función se encarga del envío de datos al máster a través del periférico UART. Se espera que el buffer datatx contenga una trama de 14 bytes con los datos que se desean transmitir.

Al igual que en la función de recepción, primero se comprueba si el UART fue inicializado correctamente. Tras esto, se llama a UARTdrv->Send() para iniciar la transmisión de la trama. La función espera hasta que se genere el evento ARM_USART_EVENT_SEND_COMPLETE, que es gestionado por el callback USART_Callback.

La función void USART_Callback (uint32_t event):

```

51 void USART_Callback(uint32_t event) {
52     if (event & ARM_USART_EVENT_RECEIVE_COMPLETE) {
53         osThreadFlagsSet(TID_UartRx, ARM_USART_EVENT_RECEIVE_COMPLETE);
54         // printf("[UART CALLBACK] Comando recibido\n");
55     }
56
57     if (event & ARM_USART_EVENT_SEND_COMPLETE) {
58         osThreadFlagsSet(TID_UartTx, ARM_USART_EVENT_SEND_COMPLETE);
59         // printf("[UART CALLBACK] Respuesta enviada al master\n\n");
60     }
61 }

```

Esta función funciona como un manejador de eventos para la comunicación UART. Será invocada automáticamente por el driver CMSIS-Driver cuando se produzcan eventos relevantes durante la transmisión o recepción de datos.

- Si se recibe un evento ARM_USART_EVENT_RECEIVE_COMPLETE, se notifica al hilo de recepción (TID_UartRx) mediante una señal (osThreadFlagsSet) para continuar con el procesamiento del comando recibido.
- Si se detecta el evento ARM_USART_EVENT_SEND_COMPLETE, se notifica al hilo de transmisión (TID_UartTx) de que los datos han sido enviados con éxito.

- **Módulo bajo consumo:**

En este módulo se implementa la gestión para introducir a la tarjeta núcleo en el modo de bajo consumo, que presenta la siguiente descripción funcional.

Bajo consumo	Descripción
Objetivo	Gestiona el bajo consumo en función del estado del pin PB1
Entradas	N/A
Salidas	N/A
Ficheros	lowmode.c, lowmode.h, stm32f4xx_it.c

Además, no implementa ningún recurso RTOS. Únicamente contiene dos funciones las cuales se usan en el código principal del slave (PRINC_SLAVE) para introducir al sistema en bajo consumo. La primera función ‘void Init_PB1_WAKEUP (void)’ que permite inicializar al pin PB1 como interrupción ya que será el pin que gestiona este modo. Este pin funciona como si de un ‘Chip Select’ se tratase, de forma que cuando se detecta el nivel bajo del PB1 permanece en bajo consumo, y con un nivel alto del PB1 estará despierta la tarjeta núcleo. Esta implementación se detallará en la parte que se refiere al principal del slave.

```
void Init_PB1_WAKEUP (void) {
    ...
    GPIO_InitTypeDef GPIO_InitStruct;
    ...
    __HAL_RCC_GPIOB_CLK_ENABLE(); // Se habilita el reloj para el GPIOB
    GPIO_InitStruct.Pin = GPIO_PIN_1;
    GPIO_InitStruct.Pull = GPIO_PULLDOWN; // No hay pull-up ni pull-down
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;

    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct); // Inicializa el pin PB1
    ...
    /* Enable and set Button EXTI Interrupt to 1 */
    HAL_NVIC_SetPriority(EXTI1 IRQn, 0x0F, 0x00);
    HAL_NVIC_EnableIRQ(EXTI1 IRQn); // Las interrupciones se manejan en el vector EXTI1
}
```

```
void EXTI1_IRQHandler (void) {
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_PIN) {
    ...
}
```

En la captura de la derecha se tiene las funciones del código dentro del archivo de interrupciones (stm32f4xx_it.c). Por otra parte, la función ‘void Enter_SleepMode (void)’ implementa el modo de bajo consumo como tal:

```
void Enter_SleepMode (void)
{
    GPIO_InitTypeDef GPIO_InitStruct;
    ...
    /* Disable USB Clock */
    __HAL_RCC_USB_OTG_FS_CLK_DISABLE();

    /* Configure all GPIO as analog to reduce current consumption on
     * enable clock */
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOD_CLK_ENABLE();
    __HAL_RCC_GPIOE_CLK_ENABLE();
    __HAL_RCC_GPIOF_CLK_ENABLE();
    __HAL_RCC_GPIOG_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOI_CLK_ENABLE();
    __HAL_RCC_GPIOJ_CLK_ENABLE();
    __HAL_RCC_GPIOK_CLK_ENABLE();

    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Pin = GPIO_PIN_All;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    ...
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
    HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
    HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
    HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);
    HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
    HAL_GPIO_Init(GPIOH, &GPIO_InitStruct);
    HAL_GPIO_Init(GPIOI, &GPIO_InitStruct);
    HAL_GPIO_Init(GPIOJ, &GPIO_InitStruct);
    HAL_GPIO_Init(GPIOK, &GPIO_InitStruct);

    /* Request to enter SLEEP mode */
    HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFI);

    /* Resume Tick interrupt if disabled prior to sleep mode entry */
    HAL_ResumeTick();
    ...
    /* Configure all GPIO as analog to reduce current consumption on
     * enable clock */
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOD_CLK_ENABLE();
    __HAL_RCC_GPIOE_CLK_ENABLE();
    __HAL_RCC_GPIOF_CLK_ENABLE();
    __HAL_RCC_GPIOG_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOI_CLK_ENABLE();
    __HAL_RCC_GPIOJ_CLK_ENABLE();
    __HAL_RCC_GPIOK_CLK_ENABLE();

    __HAL_RCC_USB_OTG_FS_CLK_ENABLE();
}
```

Primeramente, se habilitan los relojes de todos los puertos de la tarjeta, a excepción del reloj B que es el que emplea el pin que despertará a la ‘slave’ del bajo consumo, el PB1. Después se configuran en modo analógico para reducir su consumo para que en el siguiente paso se deshabiliten justo los mismos relojes que han sido habilitados anteriormente. Después se habilita el pin PB1 que gestiona el estado del bajo consumo, se suspende el tick del sistema para entrar en el bajo consumo en la función HAL_PWR_EnterSLEEPMode (PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFI); que se quedará aquí parada hasta que se vuelva a despertar a la ‘slave’ con la configuración del PB1. Posteriormente se vuelve a recuperar el tick del sistema, se habilitan los relojes deshabilitados y se habilita el puerto USB.

- **Módulo principal del slave:**

Este módulo controla el funcionamiento lógico de la placa B (slave) del invernadero inteligente. Su función principal es gestionar el estado general del sistema, el encendido y apagado de sensores según el modo de funcionamiento, el procesamiento de comandos recibidos por UART, y la recolección y transmisión de datos sensorizados a través de una cola RTOS hacia el máster.

Principal Slave	Descripción
Objetivo	Controla el funcionamiento lógico de la placa B (slave) del invernadero inteligente
Entradas	N/A
Salidas	N/A
Ficheros	slave.c, slave.h

Recurso	Valor	Descripción
Hilo	tid_slave	Hilo principal que gestiona el comportamiento del sistema slave.
Cola	mid_Slave_Com_MsgQueue	Cola que recibe los datos de las medidas de todos los sensores y ADC's

La función int Init_Slave(void):

```

67  int Init_Slave (void) {
68
69      tid_slave = osThreadNew(ThSlave, NULL, NULL);
70  □      if (tid_slave == NULL) {
71          return(-1);
72      }
73  □      Init_Slave_Com_Queue();
74
75
76      return(0);
77  }

```

Inicializa el hilo principal del esclavo y la cola de comunicación mid_Slave_Com_MsgQueue, donde se colocan las mediciones a enviar al máster. Esta función se ejecuta al arranque del sistema esclavo.

La función void ThSlave(void *argument):

```
80  □ void ThSlave (void *argument) {
81      InitializeSensors();
82      Init_PB1_WAKEUP();
83      □ while (1) {
84          processCommand(commandrx[1]);
85          □ switch (currentState) {
86
87
88          case INITIALIZE:
89              InitializeSensors();
90              break;
91
92          case WAKE_UP:
93              readGPIO_Input(currentState);
94              break;
95
96          case LOW_POWER:
97              readGPIO_Input(currentState);
98              break;
99
100         case ALL_MEAS:
101             osMessageQueueGet(mid_MsgQueueADCs, &rx_ADCs, NULL, 0u);
102             osMessageQueueGet(mid_MsgQueueBME680, &rx_BME680, NULL, 0u);
103             osMessageQueueGet(mid_MsgQueueBH1750, &rx_BH1750, NULL, 0u);
104             osMessageQueueGet(mid_MsgQueueHCSR04, &rx_HCSR04, NULL, 0u);
105
106             tx_medidas.Voltaje_RTD = rx_ADCs.RTD_Vol;
107             tx_medidas.Voltaje_CONSUMPTION = rx_ADCs.CONSUMPTION_Vol;
108             tx_medidas.humidity = rx_BME680.humidity;
109             tx_medidas.air_quality = rx_BME680.air_quality;
110             tx_medidas.luminosity = rx_BH1750.lum;
111             tx_medidas.aquaLevel = rx_HCSR04.quantity;
112
113             osMessageQueuePut(mid_Slave_Com_MsgQueue, &tx_medidas, NULL, 100);
114             break;
115         }
116     }
117 }
118 }
```

Este hilo controla todo el comportamiento del dispositivo slave mediante una máquina de estados que reacciona a los comandos recibidos por UART. El sistema responde a las siguientes órdenes:

- **INITIALIZE:** Ejecuta InitializeSensors() para configurar todos los sensores y periféricos del sistema, dejándolos listos para su uso.
- **WAKE_UP:** El sistema enciende un LED indicador para señalar que el dispositivo está despierto y en modo activo. La transición a este estado se produce al detectar una señal de nivel alto en el pin PB1 mediante readGPIO_Input().
- **LOW_POWER:** En este modo, el sistema apaga todos los sensores y desactiva la comunicación UART para reducir el consumo energético. Permanece en estado de bajo consumo hasta que el pin PB1 cambie a nivel alto, lo que lo sacará de este estado.
- **ALL_MEAS:** Recopila mediciones de todos los sensores activos, recuperando los datos a través de las colas y los prepara para su transmisión al dispositivo máster, almacenándolos en la cola mid_Slave_Com_MsgQueue.

El comportamiento del hilo depende directamente del valor de commandrx[1], que contiene el código del comando UART recibido.

La función void InitializeSensors(void):

```
120 □ void InitializeSensors (void) {
121     Init_HCSR04 ();
122     Init_ThBH1750();
123     Init_ThADCs();
124     Init_Thbme680();
125     Init_LEDs();
126 }
127 }
```

Inicializa todos los sensores utilizados en el sistema: sensor ultrasónico (HC-SR04), sensor de luz (BH1750), sensores analógicos (RTD y consumo), y sensor ambiental BME680. También inicializa los LEDs de estado.

La function void processCommand(uint8_t cmd):

```

140 void processCommand (uint8_t cmd) {
141
142 switch(cmd) {
143     case 0x01: //All meas
144         currentState = ALL_MEAS;
145         break;
146     case 0xAA: //Sleep
147         currentState = LOW_POWER;
148         break;
149     case 0xBB: //Wake-Up
150         currentState = WAKE_UP;
151         break;
152 }
153 }
```

Recibe el byte de comando (cmd) y actualiza el estado del sistema según su valor:

- 0x01 → ALL_MEAS: Recoger y enviar datos.
- 0xAA → LOW_POWER: Pasar a modo de bajo consumo.
- 0xBB → WAKE_UP: Despertar del modo bajo consumo.

La función void readGPIO_Input(StatesPrincipal_t estado):

```

155 void readGPIO_Input(StatesPrincipal_t estado){
156
157     if(estado == WAKE_UP){
158         if(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_1) == GPIO_PIN_SET){
159             HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET);
160             osDelay(10);
161         }
162     }
163     if(estado == LOW_POWER){
164         if(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_1) == GPIO_PIN_RESET){
165             HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_RESET);
166             Deinit_I2C_BME680();
167             Deinit_I2C();
168             NVIC_DisableIRQ(USART3_IRQn);
169             Enter_SleepMode();
170             NVIC_EnableIRQ(USART3_IRQn);
171             GPIO_HCSR04();
172             CONSUMPTION_pin_F429ZI_config();
173             RTD_pin_F429ZI_config();
174             Init_LEDs();
175             Init_I2C();
176             Init_I2C_MASTER();
177         }
178         osDelay(10);
179     }
180 }
```

Gestión del estado de los pines GPIO en función del estado actual:

- En **WAKE_UP**, si se detecta el pin (PB1) activo, se enciende el LED rojo (PB14).
- En **LOW_POWER**, si el pin deja de estar activo, se apagan los periféricos (sensores, UART) y se entra en modo Sleep usando Enter_SleepMode(). Luego se reconfiguran todos los periféricos tras despertar.

2. Tarjeta maestra

- Memoria EEPROM AT24C256:

Este módulo gestiona el almacenamiento y lectura de las mediciones de los sensores (temperatura, humedad, luminosidad, calidad del aire, nivel de agua, consumo y hora) utilizando una memoria EEPROM AT24C256 conectada por I2C. Los datos se organizan en zonas específicas de la memoria para permitir su persistencia.

EEPROM	Descripción
Objetivo	Almacenamiento y lectura de las mediciones de los sensores
Ficheros	AT24C256.c, AT24C256.h

La memoria AT24C256 se distribuye en bloques de 64 páginas de 64 bytes cada una, totalizando 4096 bytes. Para la organización de los datos, cada tipo de variable sensorizada ocupa una página específica. La asignación actual es la siguiente:

- TEMP_PAGE_ADDR (0x0000) – Página 0: medidas de temperatura (2 bytes por muestra)
- CONS_PAGE_ADDR (0x0040) – Página 1: medidas de consumo (2 bytes por muestra)
- LUMI_PAGE_ADDR (0x0080) – Página 2: medidas de luminosidad (3 bytes por muestra)
- AIR_PAGE_ADDR (0x00A0) – Página 3: calidad del aire (1 byte por muestra)
- HUM_PAGE_ADDR (0x0100) – Página 4: medidas de humedad (2 bytes por muestra)
- WATER_PAGE_ADDR (0x0140) – Página 5: nivel de agua (2 bytes por muestra)
- HORA_PAGE_ADDR (0x0180) – Página 6: hora exacta de la medida (3 bytes por muestra: horas, minutos y segundos)

Cada variable dispone de espacio para almacenar hasta 10 registros en su página correspondiente, de manera circular.

La función uint8_t writememory(uint8_t* WriteData, uint32_t longwrite):

```
53 uint8_t writememory( uint8_t* WriteData, uint32_t longwrite){  
54  
55     //Limpiamos los flags de eventos al realizar una nueva transmisión  
56     I2C_Event = 0U;  
57     //Transmitimos la dirección del registro donde escribir y sus datos  
58     I2Cdrv->MasterTransmit (ADDRESS_MEMORIA, WriteData, longwrite, false);  
59     //Esperemos hasta que se realice la transmisión  
60     while ((I2C_Event & ARM_I2C_EVENT_TRANSFER_DONE) == 0U);  
61     //Comprobamos que todos los datos se han transmitido  
62     if ((I2C_Event & ARM_I2C_EVENT_TRANSFER_INCOMPLETE) != 0U) return -1;  
63  
64     return 0;  
65 }
```

Esta función permite escribir datos en la memoria EEPROM AT24C256 a través del bus I2C. La dirección del esclavo (EEPROM) es fija y está definida por la constante ADDRESS_MEMORIA. El primer parámetro que se envía (WriteData) contiene tanto la dirección interna del registro como los datos que se quieren guardar. Se realiza la transmisión I2C usando MasterTransmit, sin reiniciar el bus al final. Luego, espera de forma bloqueante hasta que se reciba el evento de “transferencia completada”. Si el flag de transferencia incompleta se activa, devuelve error (-1). En caso de éxito, retorna 0.

La función uint8_t readmemory (uint8_t* Registerdirectory, uint8_t* data, uint32_t longlecture):

```

67 uint8_t readmemory(uint8_t* Registerdirectory, uint8_t* data, uint32_t longlecture){
68
69     //Dirección del registro
70     //Limpiamos los flags de eventos al realizar una nueva transmisión
71     I2C_Event = 0U;
72     //Transmitimos la dirección del registro a leer
73     I2Cdrv->MasterTransmit (ADDRESS_MEMORIA, Registerdirectory, 2, true);
74     //Esperemos hasta que se realice la transmisión
75     while ((I2C_Event & ARM_I2C_EVENT_TRANSFER_DONE) == 0U);
76     //Comprobamos que todos los datos se han transmitido
77     if ((I2C_Event & ARM_I2C_EVENT_TRANSFER_INCOMPLETE) != 0U) return -1;
78
79     //Lectura datos registro
80     //Limpiamos los flags de eventos al realizar una nueva transmisión
81     I2C_Event = 0U;
82     //Solicitamos recibir los datos
83     I2Cdrv->MasterReceive (ADDRESS_MEMORIA, data, longlecture , false);
84     //Esperemos hasta que se realice la transmisión
85     while ((I2C_Event & ARM_I2C_EVENT_TRANSFER_DONE) == 0U);
86     //Comprobamos que todos los datos se han transmitido
87     if ((I2C_Event & ARM_I2C_EVENT_TRANSFER_INCOMPLETE) != 0U) return -1;
88
89     return 0;
90 }

```

La función readmemory permite recuperar datos previamente almacenados en la EEPROM AT24C256. Utiliza dos pasos I2C separados: primero transmite la dirección interna desde donde desea leer, y luego inicia la lectura real. Como en el caso de la escritura, la dirección del esclavo está fija mediante ADDRESS_MEMORIA. Durante la primera parte, se utiliza MasterTransmit con la opción repeated start habilitada para que no se libere el bus antes de comenzar la lectura. Después, se realiza la recepción con MasterReceive, indicando el número de bytes que se desean leer. Ambas operaciones están controladas por el flag I2C_Event para asegurar que se complete la transacción correctamente. Esta función devuelve 0 en caso de éxito y -1 si alguna transferencia queda incompleta.

La función void guardar_medida (uint16_t base_addr, uint8_t* data, uint8_t data_len, uint8_t* index):

```

93 void guardar_medida(uint16_t base_addr, uint8_t* data, uint8_t data_len, uint8_t* index) {
94     // Calculamos cuántas entradas caben por página según el tamaño de los datos
95     uint8_t max_entries = 0x0A;
96
97     // Si el índice supera el límite, lo reseteamos
98     if (*index == max_entries) {
99         *index = 0;
100    }
101
102    // Dirección en memoria para esta medida
103    uint16_t addr = base_addr + (*index * data_len);
104
105    // Preparar el buffer: 2 bytes de dirección + data_len bytes de dato
106    write_buffer[0] = (addr >> 8) & 0xFF; // Dirección MSB
107    write_buffer[1] = addr & 0xFF; // Dirección LSB
108
109    for (uint8_t i = 0; i < data_len; i++) {
110        write_buffer[2 + i] = data[i]; // Copiamos el dato al buffer
111    }
112
113    writememory(write_buffer, 2 + data_len); // Dirección + dato
114    osDelay(5);
115
116    (*index)++; // Aumentamos el índice
117
118 }

```

Esta función se encarga de guardar una nueva medida sensorizada en una dirección concreta dentro de la memoria EEPROM AT24C256, utilizando una lógica de almacenamiento tipo buffer circular por página. La memoria tiene una dirección base (base_addr) para cada tipo de medida y los datos se almacenan secuencialmente dentro de esa sección hasta alcanzar un número máximo de entradas (max_entries = 10). Luego, el índice se reinicia, sobrescribiendo los datos más antiguos. La dirección real dentro de la memoria se calcula a partir de la base y el índice actual, multiplicado por el tamaño de la medida (data_len). Esta dirección se divide en dos bytes y se inserta al inicio del buffer de escritura (write_buffer), seguida por los datos a almacenar. Finalmente, se llama a la función writememory para enviar el buffer completo por I2C, y se introduce un pequeño retardo para asegurar la estabilidad de la EEPROM tras la escritura. El índice se incrementa para la siguiente escritura.

Esta función permite registrar de forma ordenada y cíclica las últimas 10 medidas de un tipo, facilitando su lectura posterior en el modo standby.

La función void leer_medida (uint16_t base_addr, uint8_t* destino, uint8_t data_len, uint8_t* read_index):

```
185 void leer_medida(uint16_t base_addr, uint8_t* destino, uint8_t data_len, uint8_t* read_index){  
186     uint8_t max_entries = 0x0A;  
187  
188     if (*read_index == max_entries) {  
189         *read_index = 0;  
190     }  
191  
192     uint16_t addr = base_addr + (*read_index * data_len);  
193  
194     uint8_t addr_buf[2];  
195     addr_buf[0] = (addr >> 8) & 0xFF;  
196     addr_buf[1] = addr & 0xFF;  
197  
198     readmemory(addr_buf, destino, data_len);  
199  
200     (*read_index)++;  
201 }  
202 }
```

La función leer_medida permite recuperar una medida previamente almacenada en la memoria EEPROM AT24C256 desde una posición concreta dentro de una página reservada para un tipo específico de dato. Utiliza una lógica de acceso secuencial con índice y comportamiento circular, permitiendo recorrer de forma ordenada las últimas 10 medidas almacenadas por tipo de sensor.

La dirección de lectura se calcula a partir de la dirección base (base_addr) de la sección correspondiente, sumando el desplazamiento del índice actual multiplicado por el tamaño de cada medida (data_len). Esta dirección se divide en dos bytes que se usan para indicar al esclavo I2C dónde comenzar la lectura.

Se llama a la función readmemory para leer los datos directamente desde la EEPROM al buffer de destino. Tras la lectura, el índice se incrementa, y si alcanza el máximo permitido (10), se reinicia a cero, implementando así un recorrido circular sobre las entradas almacenadas.

La función void agregar_buffCirc (const char* texto):

```
331 void agregar_buffCirc (const char* texto){  
332     snprintf(web_buffer[temp_read_index - 1], STRING_SIZE, "%s", texto);  
333 }
```

La función agregar_buffCirc se encarga de insertar una nueva cadena de texto formateada dentro del buffer circular de visualización web. Este buffer mantiene una representación en texto de todas las medidas de los sensores con timestamp, que luego pueden mostrarse en la interfaz web del sistema cuando se encuentra en modo standby.

Para almacenar la medida en la posición adecuada, se utiliza el índice temp_read_index - 1, ya que el índice de lectura de las medidas ha sido incrementado previamente tras recuperar los datos desde la EEPROM. Esto garantiza que el valor más reciente se almacene en la posición correspondiente del buffer circular, asegurando una presentación ordenada de las últimas medidas almacenadas.

La función void leer_una_medida_y_mostrar(void):

```
335 void leer_una_medida_y_mostrar(void) {  
336     uint16_t temp = 0, hum = 0, agua = 0, consumo = 0;  
337     uint8_t aire = 0, h = 0, m = 0, s = 0;  
338     uint32_t luz = 0;  
339  
340     leer_todas_las_medidas(&temp, &hum, &aire, &luz, &consumo, &agua, &h, &m, &s);  
341  
342 // printf("[LEIDO] Temp: %.1f, Hum: %.1f, Agua: %.1f, Luz: %.1f, Aire: %d, Hora: %02d:%02d:%02d\n",  
343 //     temp/1.0f , hum/1.0f, agua/1.0f, luz/1.0f, aire, h, m, s);  
344  
345     char linea[STRING_SIZE];  
346     snprintf(linea, STRING_SIZE,  
347             "H: %02d:%02d | T: %.2f°C | H: %.2f %% | WL: %.2f %% | L: %.2f lx | AQ: %d PM",  
348             h, m, s, temp/100.0f, hum/100.0f, agua/100.0f, luz/100.0f, aire);  
349  
350     agregar_buffCirc(linea);  
351 }
```

Esta función combina la lectura de todas las variables sensorizadas desde la EEPROM con la conversión y el formateo de sus valores a una cadena.

Primero, se definen e inicializan las variables para almacenar las medidas: temperatura, humedad, calidad del aire, luz, agua, consumo eléctrico y hora. Luego, se llama a la función leer_todas_las_medidas, que carga desde la EEPROM los valores correspondientes a una sola entrada de medición.

Después, los datos recuperados se convierten a formato flotante o entero según corresponda, y se formatean en una línea de texto estructurada con sprintf, incluyendo todos los parámetros relevantes y la hora de la medida. Finalmente, esta línea se pasa a la función agregar_buffCirc, que se encarga de almacenarla en el buffer circular, donde quedará disponible para ser mostrada en la web.

- Display LCD:

Este módulo se va a encargar de mostrar en el LCD todos los parámetros característicos durante el modo **PRESENCIAL** permitiendo simular una interacción con el usuario como si en el invernadero se tratase, permitiendo ver todos los datos en el LCD a lo largo de distintas pantallas. La descripción funcional del mismo es la siguiente:

LCD	Descripción
Objetivo	Mostrar los parámetros del invernadero en el modo presencial
Entradas	Parámetros del invernadero, flags recibidos del RFID (ENTER_INV, EXIT_INV)
Salidas	N/A
Ficheros	Thlcd.c, Thlcd.h, Arial12x12.h

Los recursos RTOS que emplea son:

Recurso	Valor	Descripción
Hilo	tid_Display	Hilo encargado del LCD

En este hilo es donde se realiza prácticamente toda la gestión de este módulo en el cuál mediante el uso de una estructura global se reciben los parámetros para ser mostrados en el LCD. Como ya se sabe, este display trabaja con el protocolo de comunicaciones **SPI**, colocado en la tarjeta mbed de aplicaciones. En la función del hilo ‘tid_Display’ se desarrolla toda la lógica, en el bucle while de la función ‘void Thread(void *arg)’. Lo primero de todo es llamar a la función ‘LCD_Initialize ()’ la cual realiza toda la gestión de inicialización del LCD:

```
void LCD_Initialize(void) {
    Init_PinesGPIO();
    LCD_Reset();
    LCD_Init();
    LCD_Clean();
    LCD_Update();
}
```

Se inicializan los pines que gestionan el LCD en la función ‘Init_PinesGPIO();’ (PA6 para el reset, PF13 para el A0, y PD14 para el CS). Posteriormente se realiza el reset en ‘LCD_Reset();’ en la cual se inicializa el bus SPI y se realiza el pequeño pulso a nivel alto de 1 us para que se inicialice correctamente. Posteriormente se llama a ‘LCD_Init();’ en la cual se inicializa el LCD con unos comandos de inicialización en los cuales no se va a entrar en detalle. Después se limpia el buffer con ‘LCD_Clean();’ y por último se actualiza gracias a la función ‘LCD_Update();’.

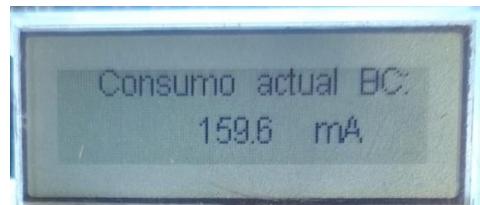
Después nos encontramos con el while, en el cual ahora se va a entrar en detalle. Se irán describiendo en orden cada pantalla que se mostrará en la aplicación real. Cuando nos encontramos en el modo STANBY el LCD lo único que mostrará será la medida realizada externamente cuando el sistema se encuentra durante el bajo consumo, ya que durante este modo no hay comunicación alguna y solo se muestra la estimación justificada más arriba durante el modo de bajo consumo:

```

/* 1º) ·Primer panel: ·modo ·bajo ·consumo.·*/
strcpy(lcd_text[0], "·Consumo ·actual ·BC:");
sprintf(lcd_text[1], "····· ·%.1f ·mA ···", 159.6f);
escrituraLCD_V2(1, lcd_text[0]);
escrituraLCD_V2(2, lcd_text[1]);
LCD_Update();

flagNFC = osThreadFlagsWait(ENTER_INV, osFlagsWaitAll, osWaitForever);

```



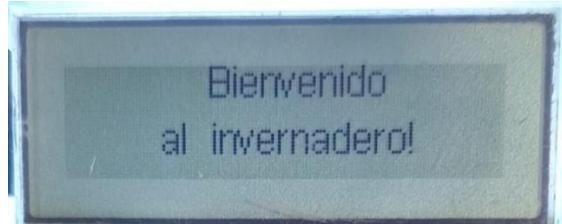
Como se puede observar al final, se espera al flag del módulo del RFID el cuál **indicará que se ha accedido al LCD de forma presencial**, lo que hará que pase a la siguiente línea de ejecución de código y cambiando el modo. Justo cuando ocurre esto, el LCD pasa a mostrar lo siguiente:

```

if((flagNFC & ENTER_INV) == ENTER_INV){
    flagNFC = 0x0000; //Flag ·reset
    ...

/* 2º) ·Segundo panel: ·NFC ·detectado.·*/
LCD_Clean();
strcpy(lcd_text[0], "····· Bienvenido ·····");
strcpy(lcd_text[1], "····· al invernadero! ·····");
escrituraLCD_V2(1, lcd_text[0]);
escrituraLCD_V2(2, lcd_text[1]);
LCD_Update();
osDelay(5000);

```



Durante cinco segundos muestra ese mensaje, permitiendo al usuario comprobar que efectivamente el acceso al invernadero ha sido correcto. Justo después de mostrar ese mensaje, entramos en el bucle del estado **PRESENCIAL** del LCD, que estará en ese bucle siempre que no se vuelva a salir del invernadero volviendo a pasar una de las tarjetas RFID. El bucle se compone de tres pantallas en la que se muestran el nivel de agua, calidad de aire, luminosidad, temperatura, humedad y consumo actual. La primera es la siguiente:

```

while(!exitLoop){
    /* 3º) ·Consumo ·actual ·de ·la ·tarjeta.·*/
    LCD_Clean();
    strcpy(lcd_text[0], "·····Consumo ·actual: ·····");
    sprintf(lcd_text[1], "····· %.1f ·mA ·····", meas_global.consumo);
    escrituraLCD_V2(1, lcd_text[0]);
    escrituraLCD_V2(2, lcd_text[1]);
    LCD_Update();

    flagNFC = osThreadFlagsWait(EXIT_INV, osFlagsWaitAll, 2500U);
    if(flagNFC == EXIT_INV){ //Devuelve 0xFFFFFFFF, ·por ·eso ·entra ba en el if porque hacia
        exitLoop = true;
        break;
    }
}

```



Como se comprueba, se muestra el consumo actual durante 2.5 segundos ya que se queda esperando durante ese tiempo por si acaso llega el flag de salida del invernadero y no tener que esperar a llegar al final en caso de que llegase entre distintas pantallas. Además, estará en ese bucle siempre que el booleano de salir del invernadero sea 'false', ya que solo adquiere el estado de 'true' cuando llega el flag de salida.

```

/* 4º) ·Primeras ·medidas.·*/
LCD_Clean();
sprintf(lcd_text[0], "L: %.1f ·lx ·w: %.1f ·%%", meas_global.lum, meas_global.quantity);
sprintf(lcd_text[1], "·CA: %d ·PM ···", meas_global.air_quality);
escrituraLCD_V2(1, lcd_text[0]);
escrituraLCD_V2(2, lcd_text[1]);
LCD_Update();

flagNFC = osThreadFlagsWait(EXIT_INV, osFlagsWaitAll, 2500);
if(flagNFC == EXIT_INV){ //Devuelve 0xFFFFFFFF, ·por ·eso ·entra ba en el if porque hacia
    exitLoop = true;
    break;
}

```



De la misma forma que el anterior, en este panel se muestran las medidas de luminosidad, nivel de agua y calidad de aire durante 2.5 segundos.

```
/* 5°) ·Segundas·medidas··*/
LCD_Clean();
sprintf(lcd_text[0], "....T: %.1f °C ....", meas_global.temperatura);
sprintf(lcd_text[1], "....H: %.1f %% ....", meas_global.humidity);
escrituraLCD_V2(1, lcd_text[0]);
escrituraLCD_V2(2, lcd_text[1]);
LCD_Update();

flagNFC = osThreadFlagsWait(EXIT_INV, osFlagsWaitAll, 2500U);
if(flagNFC == EXIT_INV){ //Devuelve 0xFFFFFFFF, por eso entra en el if porque hacia & con 0x02
    exitLoop = true;
    break;
}
```



Por último, se muestra la temperatura y humedad durante 2.5 segundos volviendo a esperar al flag de salida del invernadero en caso de que se pase. Si no llega el flag, volverá al primer panel dentro de este bucle, es decir el del consumo actual. En caso de que en algún momento llegase el flag que indica la salida del invernadero, la variable booleana ‘exitLoop’ adquirirá el valor ‘true’ lo que hará que salga del bucle pasando a mostrar el último panel que indica que se ha salido correctamente del invernadero.

```
/* 6°) ·NFC·de·salida·detectado·(mismo·ID)··*/
LCD_Clean();
strcpy(lcd_text[0], "....Hasta luego! ....");
strcpy(lcd_text[1], "....Vuelve pronto! ....");
escrituraLCD_V2(1, lcd_text[0]);
escrituraLCD_V2(2, lcd_text[1]);
LCD_Update();
osDelay(5000);
```



Una vez muestre este panel, volverá al principio del while del hilo, volviendo a mostrar constantemente el panel del consumo en bajo consumo siempre que no se reciba el flag de entrada al invernadero.

- **Lector RFID RC522:**

Este módulo se va a encargar de permitir el acceso al invernadero como si simulase el modo **PRESENCIAL** del mismo. La descripción funcional de este modo es la siguiente:

Lector RFID	Descripción
Objetivo	Permitir el acceso al invernadero en el modo PRESENCIAL
Entradas	N/A
Salidas	Booleano (insideGreenhouse) y flags (ENTER_INV, EXIT_INV) que indican salida y entrada del invernadero. SPK_ON y SPK_OFF para apagar y encender el zumbador. Flag 0x02 al hilo principal.
Ficheros	rfid.c, rfid.h

Los recursos del sistema operativo que emplea son los siguientes:

Recurso	Valor	Descripción
Hilo	TID_RFID	Hilo encargado del sensor RFID RC522
Timer	id_timMeas	Timer que espera un RFID cada segundo

La forma de acceder será pasando únicamente las dos tarjetas registradas en el sistema. Posterior a esto se encenderá una luz verde y emitirá un pitido en caso de la correcta identificación o mostrará una luz roja en caso de pasar una tarjeta no registrada y no entrará el sistema en dicho modo. Las tarjetas son del tipo MIFARE Classic 1k. Estas tarjetas son un sistema de almacenamiento que dividen su memoria en bloques. Concretamente disponen de 1024 bytes divididos en 16 sectores de 64 bytes cada uno, pero no se hará uso de ello, ya que solamente lo emplearemos con identificadores únicos de tarjeta (UID), sin tener posibilidad de editar. Este sensor se gestiona mediante el protocolo **SPI2**. La configuración de pines en el archivo 'RTE_Device.h' es la siguiente:



Como puede comprobarse se utilizan prácticamente todas las líneas de este protocolo, solo que el '*Chip Select*' (CS) es gestionado mediante GPIO, concretamente usando el **PA4**. La configuración del SPI se hace empleando la librería CMSIS Driver, cuya inicialización se hace en la función 'static void TM_MFRC522_SPI_Init(void)':

```
static void TM_MFRC522_SPI_Init (void)
{
    /* Enable the clock for the SPI2 peripheral */
    __HAL_RCC_GPIOA_CLK_ENABLE();

    /* Initialize the GPIO pins for SPI2 */
    GPIO_InitTypeDef GPIO_InitStruct;
    /* CS pin configuration */
    GPIO_InitStruct.Pin = GPIO_PIN_4;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;

    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);

    /* SPI Interface */
    SPIDriver->Initialize (SPI2_Callback);
    SPIDriver->PowerControl (ARM_POWER_FULL);
    SPIDriver->Control (ARM_SPI_MODE_MASTER | ARM_SPI_CPOL_CPHAL | ARM_SPI_MSB_LSB
                        | ARM_SPI_DATA_BITS (8), 20000000);
```

Primeramente, se realiza la configuración e inicialización del PA4 que actuará como CS para el RC522. A continuación, se inicializa el controlador SPI mediante tres funciones de la capa de abstracción CMSIS-Driver. En ‘Initialize’ se llama a la función callback que registrará las interrupciones; ‘PowerControl’ que activa el periférico; y ‘Control’ que establece los parámetros de comunicación SPI como modo maestro, polaridad y fase compatibles con el RC522 (CPOL = 1 y CPHA = 1), el de bits de MSB a LSB, 8 bits de datos y una frecuencia de 20 MHz.

Posteriormente, toda la gestión de este sensor se realiza en el hilo ‘*TID_RID*’ concretamente en su función de hilo ‘*void Th_RID(void)*’. Se va a explicar a continuación todo el proceso de detección de una tarjeta, sin entrar mucho en detalle ya que el código que implementa este módulo es bastante extenso. El objetivo principal es detectar la presencia de una tarjeta, verificar si es válida (autenticada), y actuar en consecuencia encendiendo el led RGB de la mbed y haciendo sonar el zumbador.

```
/* Initialization */
initMBED_leds();
TM_MFRC522_Init();
initTim_Meas(); //Se
take_meas(); /* Sta
```

Lo primero que aparece es la inicialización de los pines del led RGB ‘*initMBED leds ()*’, la función que inicializa este sensor ‘*TM_MFRC522_Init()*’, la función que crea el timer ‘*initTim_Meas()*’ y la función ‘*take_meas()*’ que da comienzo al timer ‘*id_timMeas*’ para esperar un flag cada segundo y realizar una lectura en caso de que hubiera para que no esté constantemente leyendo.

Posteriormente entramos dentro del bucle while, el cuál repetirá constantemente. Lo primero que nos encontramos es un wait que espera al flag ‘READID’ enviado desde el timer, como se ha comentado antes periódicamente. Una vez recibido, se llama a la función ‘*TM_MFRC522_Status TM_MFRC522_Check(uint8_t* id)*’ para verificar si hay una tarjeta presente y obtener su ID, pudiendo devolver MI_OK si se ha detectado o MI_ERR en caso de error:

```
static TM_MFRC522_Status TM_MFRC522_Check(uint8_t* id) {
    TM_MFRC522_Status status = 0;
    /* Start to find cards, return status if find or not*/
    status = TM_MFRC522_Request(PICC_REQIDL, id);
    /* If card is previously detected... */
    if(status == MI_OK) {
        /* Check for anti-collision, return card serial number 4 bytes */
        status = TM_MFRC522_AntiColl(id);
    }
    /* Send card to hibernation, ready for future operations */
    TM_MFRC522_Hiber();
    return status;
}
```

Primeramente, se llama al método ‘*TM_MFRC522_Request*’ el cual envía un comando para detectar tarjetas próximas al sensor. Si se ha devuelto MI_OK del anterior método, se ejecuta ‘*TM_MFRC522_AntiColl()*’ la cual evite posibles conflictos en caso de que se detecten varias tarjetas cerca del sensor, devolviendo el UID de la tarjeta. Finalmente se llama a ‘*TM_MFRC522_Hiber()*’ que introduce al sensor en modo bajo consumo preparado para futuras operaciones. Finalmente devuelve el último valor de la variable ‘status’ dependiendo si la lectura fue exitosa (MI_OK) o no (MI_ERR).

Volviendo al bucle while, si la lectura ha sido correcta (MI_OK) entramos dentro del if, en el cual lo primero que nos encontramos es la variable booleana ‘compare’ igualada a la función ‘*AuthID()*’ que es una función local que compara el ‘id’ obtenido de la tarjeta con id’s almacenados en un array los cuales previamente se saben a que tarjeta corresponden permitiendo detectar si la tarjeta pertenece o no al sistema. Si pertenece, devuelve ‘true’ permitiendo entrar en el siguiente if:

```

if(compare){
    osThreadFlagsSet(tid_speaker, SPK_ON);
}

if(!insideGreenhouse){
    if(estadoMaster != WEB){
        /*First time at the greenhouse: ENTER*/
        osThreadFlagsSet(tid_Display, ENTER_INV);
        ledsOFF();
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET);
        osDelay(500);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_SET); //G
        osThreadFlagsSet(TID_PRINC_MASTER, 0x20);
        insideGreenhouse = true;
    }else
        insideGreenhouse = false;

    //printf("Entrada al invernadero.\n");
}else{
    /*Check out: EXIT*/
    osThreadFlagsSet(tid_Display, EXIT_INV);
    ledsOFF();
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET); //G
    osDelay(500);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_SET); //G
    insideGreenhouse = false;
    //printf("Salida del invernadero.\n");
}
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET); //R
}

#else{
    //printf("Tarjeta erronea.\n");
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_SET); //G
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET); //R
    osDelay(500);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET); //Red
}

```

encenderse el led rojo esperando a que se vuelva a pasar una tarjeta válida.

También se baraja el caso de que se detecte una tarjeta errónea y la variable ‘status’ devuelva ‘MI_ERR’ pero no se ha implementado ninguna lógica

- Zumbador

Este módulo permite controlar el altavoz piezoelectrónico con el que cuenta la STM32 a través de un hilo. El estado de este se gestionará mediante el Timer 2 configurando a este en modo Output Compare (OC) – Toggle, con el fin de generar así una señal cuadrada de frecuencia constante sobre el pin PA0.

En lo que a la descripción del módulo se refiere, será la siguiente:

Zumbador	Descripción
Objetivo	Hacer sonar al altavoz si la entrada al invernadero ha sido correcta
Entradas	Flag para encender o apagar el altavoz (SPK_ON o SPK_OFF)
Salidas	N/A
Ficheros	speaker.c, speaker.h

Así mismo, los recursos del sistema operativo que serán necesarios emplear por el módulo serán:

Recurso	Valor	Descripción
Hilo	tid_speaker	Hilo encargado del altavoz implementado en la STM32

La función `int Init_Altavoz(void)` será aquella en la que se creará el hilo del módulo.

Este bloque if-else maneja la lógica del control de acceso del invernadero. Si el ID ha sido válido se hace sonar el zumbador de la tarjeta mbed de aplicaciones. Después entramos en el caso de que el usuario se encuentre fuera del invernadero y sea la primera vez que entre en el invernadero, mientras no esté en el estado WEB. Se notifica al hilo del LCD para mostrar la información por pantalla y se enciende el led verde indicando que se ha entrado al invernadero. Finalmente se comunica al hilo principal que se ha entrado al invernadero y pone la variable ‘insideGreenhouse’ a true. En caso de que esté en el estado WEB, se iguala a false dicha variable. Después entramos en el caso de que el usuario esté dentro del invernadero y se vuelva a pasar la tarjeta para salir del invernadero, volviendo a encender el led verde indicando que la tarjeta se ha detectado y va a salir del invernadero. En caso de que la variable ‘compare’ sea false significará que se ha detectado una tarjeta, pero no está guardada en el sistema, encendiendiendo el led rojo. También se está barajando la opción de que estemos dentro del invernadero y se introduzca una tarjeta con un identificador erróneo, por lo que volverá a

```

int Init_Altavoz (void) {

    tid_speaker = osThreadNew(speaker, NULL, NULL);
    if (tid_speaker == NULL) {
        return(-1);
    }

    return(0);
}

```

La función `'void speaker (void *argument)'` será la función principal del módulo. En esta realizaremos la correspondiente inicialización y configuración del Timer 2 y posteriormente, entraremos en el bucle while(1), donde en función del flag recibido realizaremos una secuencia de sonidos u otra.

```

void speaker (void *argument) {

    uint32_t spkFlags = 0x0000;
    initTimer2_OC ();

    while (1) {
        // Insert thread code here...

        spkFlags = osThreadFlagsWait(SPK_ON | SPK_OFF, osFlagsWaitAny, osWaitForever);

        if(spkFlags == SPK_ON) {
            correctID ();
        }else if(spkFlags == SPK_OFF) {
            wrongID ();
        }
        //osThreadYield();                                // suspend thread
    }
}

```

Finalmente, la función `'void initTimer2_OC (void)'` será quella en la que se llevará a cabo la inicialización del Timer 2 con el fin de poder generar una señal cuadrada periódica para así excitar al altavoz. En primer lugar, configurará el pin PA0 como salida alternativa con el fin de permitir así que el Timer tenga el control del pin. En segundo lugar, habilitaremos el reloj correspondiente al Timer 2. Tras esto, estableceremos los valores del Prescaler y Period para obtener así una señal de 1kHz de frecuencia. Por último, estableceremos el modo OCMODE_TOGGLE para que así se modifique el estado del pin cada vez que el contador del Timer llegue al máximo establecido.

```

void initTimer2_OC (void){

    GPIO_InitTypeDef GPIO_InitStruct;

    //Timer normal en FA
    __HAL_RCC_GPIOA_CLK_ENABLE ();

    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Alternate = GPIO_AF1_TIM2;
    GPIO_InitStruct.Pin = GPIO_PIN_0;

    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    //Timer OC
    __HAL_RCC_TIM2_CLK_ENABLE ();
    htim2.Instance = TIM2;
    htim2.Init.Prescaler = 4199;
    htim2.Init.Period = 19;
    HAL_TIM_OC_Init(&htim2);

    TIM_Channel_InitStruct.OCMode = TIM_OCMODE_TOGGLE;
    TIM_Channel_InitStruct.OCPolarity = TIM_OCPOLARITY_LOW;
    TIM_Channel_InitStruct.OCFastMode = TIM_OCFAST_DISABLE;

    HAL_TIM_OC_ConfigChannel(&htim2, &TIM_Channel_InitStruct, TIM_CHANNEL_1);
}

```

- LED RGB

Este módulo está diseñado para configurar y controlar los LEDs RGB conectados a los puertos GPIOD del microcontrolador STM32. Su funcionalidad principal es representar un nivel porcentual de agua con el que se cuenta en el depósito mediante colores, como una especie de indicador visual (tipo semáforo). En lo que a la descripción funcional del modo se refiere, será la siguiente:

Leds	Descripción
Objetivo	Encender el led RGB para mostrar la entrada al invernadero y dependiendo del nivel de agua un color u otro
Entradas	N/A
Salidas	N/A
Ficheros	LEDs.c, LEDs.h

La función '`initMBED leds(void)`' será en la que se lleve a cabo la inicialización del puerto GPIO correspondientes a los LEDs de la Mbed (Puerto D), su correspondiente configuración como salidas digitales e inicializa el estado de estos a nivel alto, ya que estos son de cátodo común y se encenderán cuando cuenten con nivel '0' lógico.

```

/* This function initialize the MBED leds*/
void initMBED_leds (void){
    GPIO_InitTypeDef GPIO_InitStruct;

    /*Enable clock to GPIOD*/
    __HAL_RCC_GPIOD_CLK_ENABLE();

    /*Set GPIOD pin */
    GPIO_InitStruct.Pin = GPIO_PIN_11 | GPIO_PIN_12 | GPIO_PIN_13;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;

    /*Init GPIOB Pins*/
    HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

    /* Common cathode. They're off with '1'*/
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, GPIO_PIN_SET); //VERDE
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET); //AZUL
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET); //ROJO
}

```

La función 'ledsON(float percentage)' será en la que en función del valor float obtenido por parámetro activará un LED y otro según el siguiente umbral definido:

Rango (%)	Color mostrado	LED activo
0.0 – 20.0	Rojo	PD13
20.1 – 70.0	Amarillo	PD13 + PD12 (Rojo + Azul)
70.1 – 99.9	Verde	PD12

```

/* This function will set a color depending on the quantity*/
void ledsON (float percentage){

    if(percentage >= 0.0f && percentage <= 20.0f){ //RED
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, GPIO_PIN_SET); //VERDE
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_SET); //AZUL
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET); //ROJO
    }

    if(percentage > 20.0f && percentage <= 70.0f){ //YELLOW
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, GPIO_PIN_SET); //VERDE
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET); //AZUL
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET); //ROJO
    }

    if(percentage > 70.0f && percentage <= 99.9f){ //GREEN
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, GPIO_PIN_SET); //VERDE
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_RESET); //AZUL
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET); //ROJO
    }
}

```

- RTC y SNTP:

El objetivo de este módulo es integrar una referencia temporal en el sistema, para así poder ser capaces de conocer la hora a la que las medidas han sido realizadas cuando nos encontramos en el estado STANDBY en la placa Máster. Para ello, emplearemos el mismo código que fue desarrollado a lo largo del Bloque Temático 1 de la asignatura con el mismo fin.

En lo que al código del RTC se refiere, este constará de las funciones principales encargadas, donde la función denominada 'void RTC_Config(void)' será la encargada tanto de habilitar como de establecer el reloj del LSE como reloj principal del RTC, como configurar todo lo relacionado a sus periféricos correspondientes.

```

// Habilitar LSE y esperar a que esté listo
RCC_OscInitTypeDef RCC_OscInitStruct = {0};
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_LSE;
RCC_OscInitStruct.LSEState = RCC_LSE_ON;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
    Error_Handler();
}

// Seleccionar LSE como fuente de reloj para el RTC
RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};
PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_RTC;
PeriphClkInitStruct.RTCClockSelection = RCC_RTCCLKSOURCE_LSE;
if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
{
    Error_Handler();
}

// Habilitar el RTC
__HAL_RCC_RTC_ENABLE();

```

55 /*#-1- Configure the RTC peripheral #####*/
56 /* Configure RTC prescaler and RTC data registers */
57 /* RTC configured as follows:
58 - Hour Format = Format 24
59 - Asynch Prediv = Value according to source clock
60 - Synch Prediv = Value according to source clock
61 - OutPut = Output Disable
62 - OutPutPolarity = High Polarity
63 - OutPutType = Open Drain */
64
65 RtcHandle.Instance = RTC; // Asignacion del periferico RTC al manejador RtcHandle
66 RtcHandle.Init.HourFormat = RTC_HOURFORMAT_24; // Configuracion en el formato
67 RtcHandle.Init.AsynchPrediv = RTC_SYNCH_PREDIV; // Se establece el valor del
68 RtcHandle.Init.SynchPrediv = RTC_SYNCH_PREDIV; // Se establece el valor del p
69 RtcHandle.Init.OutPut = RTC_OUTPUT_DISABLE; // Deshabilitacion de la señal de
70 RtcHandle.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH; // Configuracion de
71 RtcHandle.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN; // Salida en Open Drai
72 __HAL_RTC_RESET_HANDLE_STATE(&RtcHandle); // Reseteo del estado del manejador
73 if (__HAL_RTC_Init(&RtcHandle) != HAL_OK) // Se llama a la funcion HAL_RTC_Init()
74 {
75 /* Initialization Error */
76 while(1); // Error en la configuracion
77 }

La otra función importante de este módulo será la denominada 'void RTC_Alarm_Config(void)'. Dicha función será en la que se tendrá lugar la configuración de la alarma para que esta salte en los períodos deseados de tiempo.

```

void RTC_Alarm_Config(void){

    HAL_RTC_GetTime(&RtcHandle, &stimestructure, RTC_FORMAT_BIN); // Re

    alarmstrcut.AlarmTime.Hours = stimestructure.Hours; // Configuro la
    alarmstrcut.AlarmTime.Minutes = stimestructure.Minutes; // Configur

    alarmstrcut.AlarmTime.Seconds = (stimestructure.Seconds + 15) % 60;
    alarmstrcut.AlarmTime.TimeFormat = RTC_HOURFORMAT_24; // Indica que
    alarmstrcut.AlarmTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE ; /
    alarmstrcut.AlarmTime.StoreOperation = RTC_STOREOPERATION_RESET; /

    alarmstrcut.AlarmMask = RTC_ALARMMASK_HOURS
    | RTC_ALARMMASK_MINUTES
    | RTC_ALARMMASK_DATEWEEKDAY; // Si pones RTC
    // En este para
    // esta, por lo

    alarmstrcut.AlarmSubSecondMask = RTC_ALARMSUBSECONDMASK_ALL; // Co
    alarmstrcut.AlarmDateWeekDaySel = RTC_ALARMDATEWEEKDAYSEL_DATE;
    alarmstrcut.AlarmDateWeekDay = 1; // se ignora con la Mask
    alarmstrcut.Alarm = RTC_ALARM_A;

    // Configurar la alarma con interrupción
    HAL_RTC_SetAlarm_IT(&RtcHandle, &alarmstrcut, RTC_FORMAT_BIN);

    // Habilitamos la interrupcion de la alarma A y B
    HAL_NVIC_EnableIRQ(RTC_Alarm_IRQn);

}

```

Por otro lado, en lo que al código del SNTP se refiere, este contará con la función principal encargada de realizar la solicitud de la hora al servidor correspondiente, la establecerá en la estructura del RTC y configurará la alarma para que esta salte tras el periodo de tiempo indicado.

```

void time_callback(uint32_t seconds, uint32_t seconds_fraction){

if(seconds != 0){

    time_t Segundos_RTC = (time_t) seconds; // Cambio el formato de la varia
    // transcurridos desde 1970 obte
    // time_t, para asi luego pasars
    // localtime

    SNTP_Date_Hour = localtime(&Segundos_RTC); // Obtenemos los segundos de
    // y los almacenamos en la e

    /*##-1- Configure the Date of the SNTP #####
    sdatestructure.Year = SNTP_Date_Hour.tm_year - 100; // Restamos 100 al v
    // en la estructura
    // se representan des
    // RTC_Hora_Fecha su

    sdatestructure.Month = SNTP_Date_Hour.tm_mon + 1; // Sumamos 1 al valor
    // enero corresponde a

    sdatestructure.Date = SNTP_Date_Hour.tm_mday; // Valor del dia a configu
    sdatestructure.WeekDay = SNTP_Date_Hour.tm_wday; // Macro en la capa HA

    if(HAL_RTC_SetDate(&RtcHandle, &sdatestructure, RTC_FORMAT_BIN) != HAL_OK)
    {

        /* Initialization Error */
        while(1); // Error en la configuracion
    }

    /*##-2- Configure the Time of the SNTP #####
    stimestructure.Hours = (SNTP_Date_Hour.tm_hour + 2) % 24; // Establece el v
    stimestructure.Minutes = SNTP_Date_Hour.tm_min; // Establece el valor de 10
    stimestructure.Seconds = SNTP_Date_Hour.tm_sec; // Establece el valor de 10
    stimestructure.TimeFormat = RTC_HOURFORMAT_24; // Indica que el formato es
    stimestructure.DayLightSaving = RTC_DAYLIGHTSAVING_NONE ; // Desactiva el d
    stimestructure.StoreOperation = RTC_STOREOPERATION_RESET; // No se almacen

    if (HAL_RTC_SetTime(&RtcHandle, &stimestructure, RTC_FORMAT_BIN) != HAL_OK)
    {
        /* Initialization Error */
        while(1); // Error en la configuracion
    }

    /*##-3- Writes a data in a RTC Backup data Register1 #####
    HAL_RTCEX_BKUPWrite(&RtcHandle, RTC_BKP_DR1, 0x32F2); // Escribe en el res
}
}

```

- Comunicación Máster

El objetivo de este módulo es gestionar la comunicación UART desde la placa A (Máster) hacia la placa B (Slave). Esta comunicación está orientada al envío de comandos estructurados de 3 bytes desde el Máster, y a la recepción y procesamiento de respuestas de 14 bytes por parte del Slave. Las respuestas incluyen medidas de sensores o códigos de control.

Comunicación Slave	Descripción
Objetivo	Comunicación UART entre la placa A y la placa B
Entradas	Trama de 14 bytes proveniente desde la placa B
Salidas	Trama de respuesta de 3 bytes desde la placa A
Ficheros	uart.c, uart.h

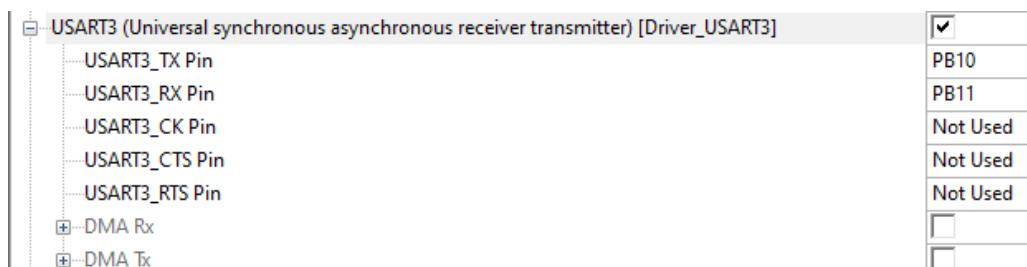
Entradas UART:

- Comandos de 3 bytes generados internamente en la Placa A, con estructura fija:
 - Byte 1: Inicio de trama (0x7E)
 - Byte 2: Código de comando (0x01, 0xAA, 0xBB)
 - Byte 3: Fin de trama (0x7F)

Salidas UART:

- Recepción de trama de 14 bytes desde la Placa B (Slave), con estructura:
 - Byte 0: Inicio de trama (0x7E)
 - Bytes 1-12: Datos codificados de sensores
 - Byte 13: Fin de trama (0x7F)
 - O, en caso de error: primer byte igual a ERROR_TRAMA

El modo de funcionamiento de esta comunicación es mediante el uso del protocolo UART. Para ello, emplearemos el pin PB10 como línea de transmisión (TX) y el pin PB11 como línea de recepción (RX) de la comunicación, tal y como se muestra a continuación.



Recurso	Valor	Descripción
Hilo	TID_UartTx	Hilo encargado de enviar comandos
Hilo	TID_UartRx	Hilo encargado de recibir y procesar datos

Para garantizar el funcionamiento correcto y eficiente, la implementación utiliza dos hilos independientes:

- **UartRx:** Escucha constantemente por comandos UART. Valida la trama recibida (inicio y fin correctos). Si la trama es válida, procede a procesar los datos recibidos.
- **UartTx:** Su función principal es gestionar la construcción de tramas de 3 bytes según el comando requerido por el sistema y garantizar su envío correcto.

La función void USART_Callback(uint32_t event):

```
27 void USART_Callback(uint32_t event) {
28     if (event & ARM_USART_EVENT_SEND_COMPLETE){
29         // printf("[UART CALLBACK] Datos enviados\n");
30         osThreadFlagsSet(TID_UartTx, ARM_USART_EVENT_SEND_COMPLETE);
31     }
32     if (event & ARM_USART_EVENT_RECEIVE_COMPLETE){
33         // printf("[UART CALLBACK] Datos recibidos correctamente\n");
34         osThreadFlagsSet(TID_UartRx, ARM_USART_EVENT_RECEIVE_COMPLETE);
35     }
36 }
37 }
```

Es la función callback asociada al controlador UART. Se ejecuta automáticamente cuando ocurre un evento relevante.

- Si el evento es ARM_USART_EVENT_SEND_COMPLETE, se activa el flag del hilo de transmisión (TID_UartTx) indicando que los datos fueron enviados correctamente.
- Si es ARM_USART_EVENT_RECEIVE_COMPLETE, se activa el flag del hilo de recepción (TID_UartRx) para procesar la respuesta.

Este mecanismo evita el bloqueo activo y sincroniza los hilos de forma eficiente.

La función void UART_Init(void):

```
39 void UART_Init(void) {
40     // 1. Inicialización
41     int32_t status;
42     status = UARTdrv->Initialize(USART_Callback);
43     if(status != ARM_DRIVER_OK) {
44         printf("[UART] Error en Initialize: %d\n", status);
45         return;
46     }
47     // 2. Encender el periférico
48     status = UARTdrv->PowerControl(ARM_POWER_FULL);
49     if(status != ARM_DRIVER_OK) {
50         printf("[UART] Error en PowerControl: %d\n", status);
51         return;
52     }
53     // 3. Configurar parámetros UART
54     status = UARTdrv->Control(
55         ARM_USART_MODE_ASYNCHRONOUS | // Modo UART (asíncrono)
56         ARM_USART_DATA_BITS_8 | // 8 bits de datos
57         ARM_USART_PARITY_NONE | // Sin paridad
58         ARM_USART_STOP_BITS_1 | // 1 bit de parada
59         ARM_USART_FLOW_CONTROL_NONE, // Sin control de flujo
60         9600 // Baudrate
61     );
62 }
```

Configura e inicializa el periférico UART:

1. Inicialización: Llama a Initialize() con el callback definido.
2. Alimentación: Activa el periférico mediante PowerControl(ARM_POWER_FULL).
3. Configuración: Define los parámetros UART: modo asíncrono, 8 bits de datos, sin paridad, 1 bit de parada, sin control de flujo, a 9600 baudios.
4. Habilitación: Activa la transmisión (TX) y la recepción (RX).

Si todo se ejecuta correctamente, se marca como inicializado (uart_initialized = true), lo que permitirá usar las funciones de transmisión y recepción más adelante.

La función int32_t UART_SendCommand(uint8_t* command):

```
84 int32_t UART_SendCommand(uint8_t* command) {
85     if (!uart_initialized) {
86         //     printf("[UART] Error: No inicializado antes de enviar\n");
87         return ARM_DRIVER_ERROR;
88     }
89     //     printf("[UART] Enviando comando: 0x%02X\n", command);
90     int32_t result = UARTdrv->Send(command, 3);
91     if(result != ARM_DRIVER_OK) {
92         //     printf("[UART] Error al enviar: %d\n", command);
93         return result;
94     }
95     osThreadFlagsWait(ARM_USART_EVENT_SEND_COMPLETE, osFlagsWaitAny, osWaitForever);
96
97     return ARM_DRIVER_OK;
98 }
```

```
63     if(status == ARM_DRIVER_OK) {
64         printf("[UART] Inicializado correctamente a 9600 baudios\n");
65         uart_initialized = true;
66     } else {
67         //     printf("[UART] Error en Control: %d\n", status);
68         // 4. Habilitar TX y RX
69         status = UARTdrv->Control(ARM_USART_CONTROL_TX, 1);
70         if (status != ARM_DRIVER_OK) {
71             //     printf("[UART] Error al habilitar TX: %d\n", status);
72             return;
73         }
74
75         status = UARTdrv->Control(ARM_USART_CONTROL_RX, 1);
76         if (status != ARM_DRIVER_OK) {
77             //     printf("[UART] Error al habilitar RX: %d\n", status);
78             return;
79         }
80     }
81 }
```

Función que transmite una trama de 3 bytes al esclavo:

- Primero verifica que el UART esté correctamente inicializado.
- Llama a UARTdrv->Send(command, 3) para lanzar la transmisión.
- Luego, espera a que el evento ARM_USART_EVENT_SEND_COMPLETE sea recibido, lo que confirma el envío exitoso.

Este mecanismo asegura que el hilo de transmisión no siga hasta haber completado la operación.

La función int32_t UART_ReceiveData(uint8_t* data):

```
105 int32_t UART_ReceiveData(uint8_t* data) {
106     if(!uart_initialized) {
107         //     printf("[UART] Error: No inicializado antes de recibir\n");
108         return ARM_DRIVER_ERROR;
109     }
110
111     return UARTdrv->Receive(data, 14);
112 }
```

Inicia una recepción de 14 bytes desde el esclavo:

- Verifica que el UART esté inicializado.
- Llama a UARTdrv->Receive(data, 14) para comenzar la recepción.
- La función no bloquea; será el callback quien indique al hilo de recepción que los datos han llegado.

La función void procesar_trama (void):

```
190 void procesar_trama (void){  
191     // 1. Procesar comandos  
192     if(comando[1]==0x01){  
193         //////////// Temperatura  
194         uint16_t rtd_scaled = ((uint16_t)data_rx[0] << 8) | data_rx[1];  
195         meas_global.temperatura = ((float)rtd_scaled) / 100.0f; // Te da el voltaje original con decimales  
196         printf("[Comando 0x01] Temperatura: 0x%02X %0%02X °C(Hexadecimal) || Temperatura: %0.1f °C \n", data_rx[0],data_rx[1],meas_global.temperatura);  
197  
198         //////////// Consumo  
199         uint16_t consumption_scaled = ((uint16_t)data_rx[2] << 8) | data_rx[3];  
200         meas_global.consumo = ((float)consumption_scaled) / 100.0f/0.004f; // Te da el consumo en mA original con decimales  
201         printf("[Comando 0x01] Consumo: 0x%02X %0%02X V(Hexadecimal) || Consumo: %0.2f mA\n", data_rx[2],data_rx[3],meas_global.consumo);  
202  
203         //////////// Luminosidad  
204         uint32_t luz_100x = (data_rx[4] << 16) | (data_rx[5] << 8) | data_rx[6];  
205         meas_global.lum = (float)luz_100x / 100.0f;  
206         printf("[Comando 0x01] Luminosidad: 0x%02X %0%02X %0%02X lx(Hexadecimal) || Luminosidad: %2f lx\n", data_rx[4],data_rx[5],data_rx[6],meas_global.lum);  
207  
208         //////////// Calidad del aire  
209         meas_global.air_quality = (int)data_rx[7];  
210         printf("[Comando 0x01] Calidad de aire: 0x%02X (Hexadecimal) || Calidad de aire: %d IAQ\n",data_rx[7],meas_global.air_quality);  
211  
212         //////////// Humedad  
213         uint16_t hum_scaled = ((uint16_t)data_rx[8] << 8) | data_rx[9];  
214         meas_global.humidity = (float)hum_scaled/ 100.0f;  
215         printf("[Comando 0x01] Humedad: 0x%02X %0%02X || Humedad: %2f %%\n", data_rx[8],data_rx[9],meas_global.humidity);  
216  
217         //////////// Nivel de agua  
218         uint16_t nivel_agua_recibido = ((uint16_t)data_rx[10] << 8) | (uint16_t)data_rx[11];  
219         meas_global.quantity = (float)nivel_agua_recibido / 100.0; // Convertir a porcentaje  
220         printf("[Comando 0x01] Nivel de agua: %02X %0%02X %%| Nivel de agua: %0.2f %%\n", data_rx[10],data_rx[11], meas_global.quantity);  
221  
222         //#[EEPROM] Guardar en la memoria solo en el modo STANDBY  
223         if(estadoMaster == STANDBY){  
224             guardar_temperatura(rtd_scaled);  
225             guardar_humedad(hum_scaled);  
226             guardar_luminosidad(luz_100x);  
227             guardar_calidad_aire(meas_global.air_quality);  
228             guardar_nivel_agua(nivel_agua_recibido);  
229  
230             RTC_Hora_Fecha(aShowTime, aShowDate);  
231  
232             hh = stimestructure.Hours;  
233             mm = stimestructure.Minutes;  
234             ss = stimestructure.Seconds;  
235             guardar_hora(hh, mm, ss);  
236         }  
237     }  
238     //////////// Flag para hilo principal_master avisando de que ya recibí los datos y han sido procesados  
239     osThreadFlagsSet(TID_PRINC_MASTER, comando[1]);  
240  
241     }else if(comando[1]==0xAA){  
242         if(data_rx[11]==0xAA){  
243             osThreadFlagsSet(TID_PRINC_MASTER, 0x08);  
244             printf("[Comando %02X]ACK de confirmacion dormir al SLAVE\n\n", comando[1]);  
245         }  
246  
247     }else if(comando[1]==0xBB){  
248         if(data_rx[11]==0xBB){  
249             osThreadFlagsSet(TID_PRINC_MASTER, 0x80);  
250             printf("[Comando %02X]ACK de confirmacion despertar al SLAVE\n\n", comando[1]);  
251         }  
252     }  
}
```

Se encarga de interpretar y procesar los datos recibidos por UART desde la placa B(Slave). Utiliza la variable global rx_buffer [] (de 14 bytes) como buffer de entrada y actualiza la estructura global meas_global con las medidas extraídas. El comportamiento de esta función varía en función del comando enviado (comando [1]), permitiendo tanto la lectura de sensores como el control de estado del sistema de la placa B(Slave).

Cuando el comando es 0x01, la función procesa una trama con medidas de sensores. Para cada variable, extrae los bytes correspondientes, los convierte al formato correcto y los escala según el factor definido para obtener el valor real:

- **Temperatura:** Se forma un entero de 16 bits a partir de data_rx[0] y data_rx[1], y se divide entre 100 para obtener la temperatura en grados.
- **Consumo eléctrico:** A partir de data_rx[2] y data_rx[3], se calcula el voltaje dividido por 0.004, obteniendo así el consumo en mA.
- **Luminosidad:** Se utilizan tres bytes (data_rx[4], [5], [6]) para formar un entero de 24 bits que se divide entre 100, resultando en lux.
- **Calidad del aire:** El byte data_rx[7] representa directamente el índice IAQ.
- **Humedad relativa:** Los bytes data_rx[8] y [9] forman un entero de 16 bits que se divide entre 100 para obtener el valor porcentual.
- **Nivel de agua:** Se reconstruyen los bytes data_rx[10] y [11] como un valor en porcentaje de llenado del tanque de agua.

Si el sistema se encuentra en el estado STANBY, los datos se almacenan en la memoria EEPROM mediante funciones dedicadas. También se guarda la hora actual utilizando el RTC, asegurando así un registro histórico completo.

Una vez procesada la trama, se notifica al hilo principal del máster (TID_PRINC_MASTER) mediante una señal RTOS. Esto permite sincronizar la ejecución del hilo principal con la recepción y tratamiento de los datos. Por otro lado, la función también contempla el procesamiento de comandos especiales:

- Si comando [1] == 0xAA, se trata de una solicitud para poner al esclavo en modo de bajo consumo. Si el byte de confirmación (data_rx[11]) es también 0xAA, se interpreta como un ACK.
- Si comando [1] == 0xBB, corresponde a una orden para despertar al esclavo. Si data_rx[11] == 0xBB, se interpreta igualmente como una confirmación.

- Principal Máster

Principal Máster	Descripción
Objetivo	Controla el funcionamiento lógico de la placa A (MÁSTER) del invernadero inteligente
Entradas	N/A
Salidas	N/A
Ficheros	Principal_Master.c, Principal_Master.h

Recurso	Valor	Descripción
Hilo	principal_Master	Hilo principal que gestiona el comportamiento del sistema MÁSTER.

El objetivo del fichero Principal_Master.c es centralizar el control del invernadero sobre la placa MÁSTER. Se encarga de gestionar el comportamiento del invernadero mediante un autómata de estados, controlando la comunicación UART con la placa SLAVE, coordinando periféricos locales como el RFID y la memoria EEPROM, y actualizando los datos para que se puedan visualizar en la Web.

El sistema implementa un autómata de cuatro estados:

- INIT: inicialización completa del sistema.
- STANBY: modo de espera de bajo consumo.
- PRESENCIAL: funcionamiento presencial con detección de usuario.
- WEB: funcionamiento remoto vía conexión web.

Se ha utilizado una máquina de estados para organizar el comportamiento del sistema de manera estructurada y clara. Esto permite que cada estado tenga un conjunto bien definido de acciones y condiciones de transición, facilitando el mantenimiento y la escalabilidad del código.

Cada estado ejecuta una serie de acciones específicas:

- INIT: se configuran los periféricos necesarios y se sincroniza el reloj.
- STANBY: se apagan los módulos no necesarios, se espera una alarma o evento para actuar.
- PRESENCIAL: se solicita una medida a la placa SLAVE si se detecta un usuario dentro del invernadero.
- WEB: se atienden peticiones remotas vía conexión web.

Las transiciones se activan por condiciones lógicas como el valor de insideGreenhouse, web, y flags generados por la alarma del RTC, el RFID o la WEB. A continuación, se explicará el funcionamiento de cada estado:

Durante el estado INIT, se inicializan los siguientes módulos:

- Comunicación UART
- Bus I²C
- Reloj en tiempo real (RTC)
- SNTP para sincronización horaria
- Lector RFID

- Memoria EEPROM AT24C256
- LEDs
- Temporizadores

Se introduce un osDelay(5000) tras la inicialización para asegurar que todos los periféricos se configuren correctamente y permitir la recepción de la hora desde el servidor SNTP. Tras la inicialización, el sistema pone la placa SLAVE en suspensión para optimizar el consumo energético antes de entrar en el estado STANBY.

```

    case INIT:
        UART_Init();
        I2C_Initialize();
        RTC_Config();
        init_despertar_Slave();
        Init_Altavoz();
        Init_RFID();
        initMBED_leds();
        Init_Timers();
        init_SNTP();

        osDelay(5000); //Tiempo para que se pueda inicializar
        printf("INVERT TECH se ha configurado correctamente\n\r");
        dormir_Slave();
        estadoMaster = STANBY;
        break;
    
```

Durante el estado de STANDBY, el sistema espera pasivamente una de tres flags:

- 0x10: activación periódica para realizar una medición automática, mediante la alarma del RTC.
- 0x20: presencia detectada (insideGreenhouse == 1), cambio de estado a PRESENCIAL.
- 0x40: conexión web activa (web == 1), cambio de estado a WEB.

Cuando se recibe el flag 0x10, proveniente de la alarma, se procede a despertar al SLAVE, utilizando la función despertar_Slave(). Se actualiza el comando que se le enviará al SLAVE para que responda con todas las medidas de los sensores “comando[1] = MEAS”, y se envía un flag al hilo que gestiona la comunicación para que envié la información actualizada al SLAVE, y se queda esperando confirmación de que se reciben las medidas del SLAVE. Una vez el SLAVE envíe las medidas, se recibe la confirmación y se duerme el SLAVE con la función dormir_Slave().

En los dos otros dos flags, se realiza la misma secuencia. Se despierta al SLAVE, se desactiva la alarma y se cambia al estado correspondiente dependiendo la condición que se reciba.

```

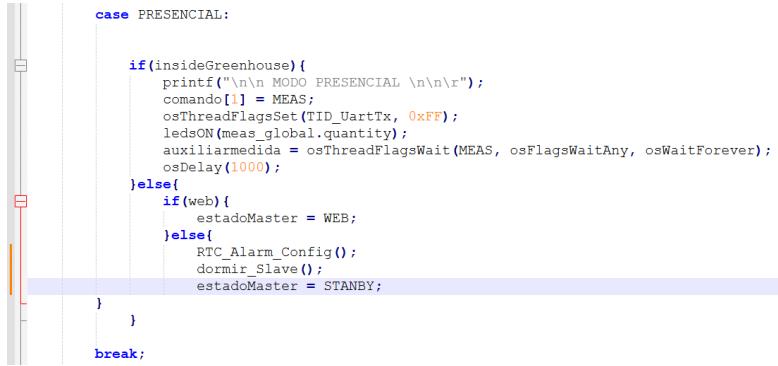
    case STANBY:
        // La Web y el modo Presencial estan inactivos, la placa Slave se duerme
        ledsOFF();

        uint32_t flagAlarma = osThreadFlagsWait(0x70, osFlagsWaitAny, osWaitForever);
        // Esperar Alarma cada x tiempo
        if(flagAlarma == 0x10){
            RTC_Alarm_Config();
            despertar_Slave();
            comando[1] = MEAS;
            osThreadFlagsSet(TID_UartTx, 0xFF);

            // Esperar confirmacion
            auxiliamedida = osThreadFlagsWait(MEAS, osFlagsWaitAny, osWaitForever);
            printf("Las medidas se han recibido correctamente\n\r");
            dormir_Slave();
        }else if(flagAlarma == 0x20 && insideGreenhouse == '1' && web == '0'){
            //auxxd=true;
            despertar_Slave();
            HAL_RTC_DeactivateAlarm(&RtcHandle, RTC_ALARM_A);
            estadoMaster = PRESENCIAL;
        }else if(flagAlarma == 0x40 && web == '1' && insideGreenhouse == '0'){
            //auxxd=true;
            despertar_Slave();
            HAL_RTC_DeactivateAlarm(&RtcHandle, RTC_ALARM_A);
            estadoMaster = WEB;
        }
    
```

Este diseño equilibra automatización, interacción humana y control remoto, adaptándose perfectamente a las necesidades de un invernadero inteligente. Se prioriza el ahorro energético, la reactividad ante eventos y la flexibilidad de operación, pilares fundamentales en la agricultura de precisión.

El estado PRESENCIAL, se activa cuando se detecta que hay alguien dentro del invernadero (insideGreenhouse == 1). Se solicita al SLAVE la toma de medidas ambientales mediante el comando MEAS. Una vez recibidas, se podrán visualizar en el LCD. Si el usuario abandona el invernadero (insideGreenhouse == 0) y no hay conexión web, el sistema vuelve al estado STANBY. Si en cambio web == 1, se transita al estado WEB.



Se entra en este estado WEB, cuando hay conexión web activa (web == 1). Se solicita al SLAVE que se prepare para el modo remoto mediante el comando INIT_WEB, y se espera confirmación, recibiendo a su vez las medidas del SLAVE, que se actualizarán en la web. Cuando la conexión web se interrumpe (web == 0), el sistema evalúa si debe pasar a STANBY (si no hay presencia) o a PRESENCIAL (si hay un usuario dentro del invernadero).



La comunicación se realiza mediante UART, utilizando un protocolo de tramas de 3 bytes: [0x7E, CMD, 0x7F], donde CMD puede ser WAKE_UP, MEAS, SLEEP o INIT_WEB.

```

/** MACROS PARA EL COMANDO */
#define SLEEP          0xAA
#define WAKE_UP        0xBB
#define INIT_WEB       0x01
#define MEAS           0x01

```

Se emplean las funciones del sistema operativo CMSIS RTOS2 para sincronizar el envío y recepción de tramas mediante flags (osThreadFlagsSet, osThreadFlagsWait). Esto asegura que la comunicación se complete correctamente antes de continuar la ejecución.

El control de encendido y apagado del SLAVE se lleva a cabo con las funciones despertar_Slave() y dormir_Slave(). La función despertar_Slave() activa el pin GPIO PB1, utilizado para encender la placa

SLAVE. Posteriormente, se envía el comando WAKE_UP vía UART y se espera confirmación. La función dormir_Slave() realiza el proceso inverso, enviando el comando SLEEP y apagando el pin GPIO. Este mecanismo dual (hardware y software) garantiza sincronización precisa y minimiza el consumo eléctrico del SLAVE.

```
void despertar_Slave(void){  
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);  
    osDelay(5000);  
    comando[1] = WAKE_UP;  
    osThreadFlagsSet(TID_UartTx, 0xFF);  
    auxiliar_despertar = osThreadFlagsWait(0x80, osFlagsWaitAll, osWaitForever);  
}  
  
static uint32_t dormir_Slave(void){  
    uint32_t aux;  
    comando[1] = SLEEP;  
    osThreadFlagsSet(TID_UartTx, 0xFF);  
    aux = osThreadFlagsWait(0x08, osFlagsWaitAll, osWaitForever);  
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);  
    return aux;  
}
```

El sistema utiliza el módulo RTC para activar alarmas periódicas que controlan cuándo deben tomarse mediciones. En la inicialización, se sincroniza la hora local con un servidor SNTP, lo que permite registrar medidas con marca temporal precisa y útil para trazabilidad.

5 DEPURACION Y TEST.

5.1 Pruebas realizadas software.

MÓDULO RFID

1º) El lector RFID detecta una tarjeta

El objetivo de esta prueba es que, tras conseguir implementar todas las funciones, el lector RFID consiga detectar una de las tarjetas disponibles que venían con el lector. Para ello se realizará toda la inicialización en un hilo, colocando el punto de ruptura en la función ‘TM_MFRC522_Check ()’ ya de esa forma sabremos si lo ha leído.

Condiciones de entrada:

- Inicialización correcta del hilo que lo implementa
- Correcta inicialización del bus SPI
- Conexión correcta de los pines del lector RFID RC522
- El lector detecta una tarjeta sea cual sea su UID

La prueba se da como exitosa ya que en todo momento se paraba en el punto de ruptura.

2º) El lector RFID distingue las tarjetas que se pasan por él y leyendo sus identificadores

Tras la prueba anterior, el objetivo es añadir una lógica que permita distinguir los UIDs de las tarjetas y poder leerlos en la ventana ‘Watch Window’. Para ello, previamente se utiliza la aplicación de ‘NFC Tools’ que lee las tarjetas Mifare y devuelve su UID, para ponerlo en el código y al pasar la tarjeta poder comparar el UID que se pasa con el almacenado en el sistema.

Condiciones de entrada:

- Inicialización correcta del hilo que lo implementa
- Correcta inicialización del bus SPI
- Conexión correcta de los pines del lector RFID RC522
- Distinción correcta de la tarjeta pasada por el lector

Tras esto, la prueba fue considerada como exitosa ya que se leían correctamente los UIDs.

3º) Se enciende el led RGB dependiendo de si la tarjeta está o no en el sistema

De la misma forma que la prueba anterior, si la tarjeta pasada por el lector coincide con las almacenadas en el sistema emitirá una luz verde por el led RGB mientras que si la tarjeta pasada tiene un UID que no está en el sistema emitirá una luz roja.

Condiciones de entrada:

- Inicialización correcta de los pines correspondientes al led RGB de la mbed.
- Emite el color correspondiente dependiendo del UID.

Tras esto, la prueba se ha considerado exitosa ya que emite distintos colores.

4º) El zumbador de la tarjeta mbed emite un pitido

Para finalizar este módulo, al pasar una tarjeta y el UID que contiene está en el sistema, emitirá un pitido el zumbador de la tarjeta mbed de aplicaciones.

Condiciones de entrada:

- Inicialización correcta del hilo que lo implementa.
- Emite el pitido correctamente tras pasar la tarjeta.

Prueba considerada como exitosa ya que emite el pitido correctamente.

MÓDULO LCD

El objetivo de esta prueba es comprobar toda la funcionalidad del lcd en este proyecto. Para ello y como se ha visto anteriormente, se desea observar el intercambio entre cinco posibles pantallas dinámicas, todo ello dentro del modo PRESENCIAL. Las pantallas que muestran al usuario el texto que indica la entrada o la salida al invernadero tienen un periodo de tiempo de cinco segundos, mientras que las que muestran información de los parámetros tienen una duración de dos segundos y medio segundos.

Condiciones de entrada:

- Inicialización correcta del hilo
- Inicialización correcta del bus SPI
- Conexión correcta de los pines que conectan la mbed con la STM32
- Los cambios entre pantallas se realizan correctamente con los tiempos indicados
- La información mostrada es correcta

La prueba se da como exitosa al ver en el LCD toda la información requerida.

MEMORIA EEPROM AT24C256

Durante la fase de validación del sistema, uno de los puntos clave fue comprobar que la memoria EEPROM AT24C256 funcionaba correctamente como módulo de almacenamiento de medidas.

Para verificar su correcto funcionamiento, fue necesario asegurar varios aspectos: la inicialización correcta del bus I²C, la configuración del controlador, la fiabilidad de las funciones de escritura y lectura, y la coherencia de los datos almacenados.

Se implementaron funciones específicas para almacenar cada tipo de medida (temperatura, humedad, calidad del aire, luminosidad, consumo, nivel de agua y hora). Estas funciones se apoyan en una función común llamada **guardar_medida**, que organiza los datos en páginas de memoria y gestiona internamente el índice de escritura. Para la lectura se utilizó una lógica similar mediante la función **leer_medida**. Se desarrolló una prueba funcional completa que genera datos aleatorios, los guarda en la memoria y luego los recupera para compararlos byte a byte. Esta verificación permitió detectar posibles errores en la escritura o en la recuperación de la información. La comprobación de los resultados se realizó directamente desde Keil uVision, monitorizando las variables en la ventana Watches.

También se usó una función adicional para reconstruir en formato de texto cada conjunto de medidas leídas desde la EEPROM, facilitando así su visualización en la interfaz de la web del proyecto cuando se está en modo *standby*.

Gracias a estas pruebas se pudo confirmar que:

- La comunicación I²C con la memoria es estable y no presenta errores.
- La escritura y lectura de datos se realiza correctamente.
- El uso de índices circulares por tipo de medida permite organizar el espacio disponible.
- El sistema puede almacenar y recuperar medidas de forma fiable.

SENSOR BME680

El objetivo es validar el correcto funcionamiento del sensor ambiental BME680, incluyendo la inicialización I²C, la configuración del sensor, la adquisición de medidas de humedad y gas, y el cálculo del índice IAQ.

Condiciones de entrada

- Conexión del sensor BME680 a la interfaz I²C de la STM32 configurada como maestro.
- Fuente de alimentación estable y activa para el sensor.
- Definición de los parámetros de calibración extraídos directamente del sensor.
- Inicialización correcta del sistema operativo RTOS y sus recursos (hilo, temporizador, cola de mensajes).

El método de validación del módulo del sensor BME680 consiste en una inicialización controlada del sensor seguida de su configuración, ejecución periódica de mediciones ambientales (humedad y gas) y evaluación del índice de calidad del aire (IAQ). La prueba se realiza mediante un hilo (ThBME680) sincronizado con un temporizador de 1 segundo que lanza mediciones a intervalos regulares.

Durante la ejecución, el sensor responde correctamente a la lectura de su identificador (chip ID = 0x61), lo que confirma su correcta conexión y comunicación. Además, se verificó la adquisición periódica de medidas de humedad y gas, y el cálculo del índice IAQ, el cual se clasifica conforme a la escala esperada. Por tanto, el test se considera correcto.

SENSOR HC-SR04

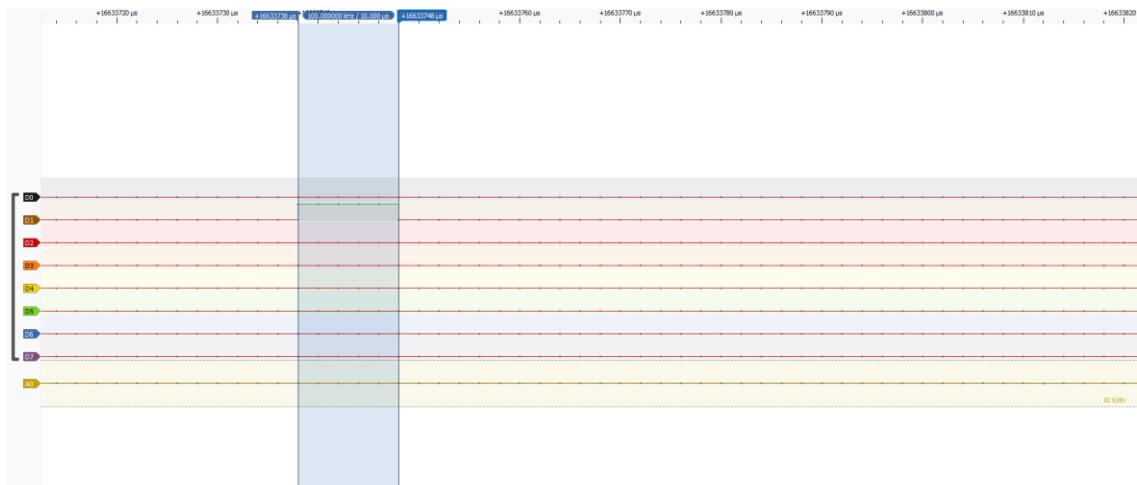
A lo largo del desarrollo de este sensor se realizaron gran cantidad de pruebas con objeto a comprobar el correcto funcionamiento y la correcta implementación de este. Dado a que el uso que se le da a este sensor no es el típico como para la mayoría de las aplicaciones, requirió diseñar pruebas específicas paso a paso para conseguir su implementación. Las pruebas son las siguientes:

1º) Sensor emite pulso de 10 µs para el pin ‘TRIGGER’:

Dado a que este sensor necesita un primer pulso de 10 µs en el pin trigger para poder emitir las ondas ultrasónicas, el objetivo es testear la producción de ese pulso del propio pin que sale de la tarjeta STM32F429ZI y que va a producir ese pulso, concretamente el pin PB2. Los módulos que permiten compr

Condiciones de entrada:

- Correcta inicialización del pin
- Correcta inicialización del hilo



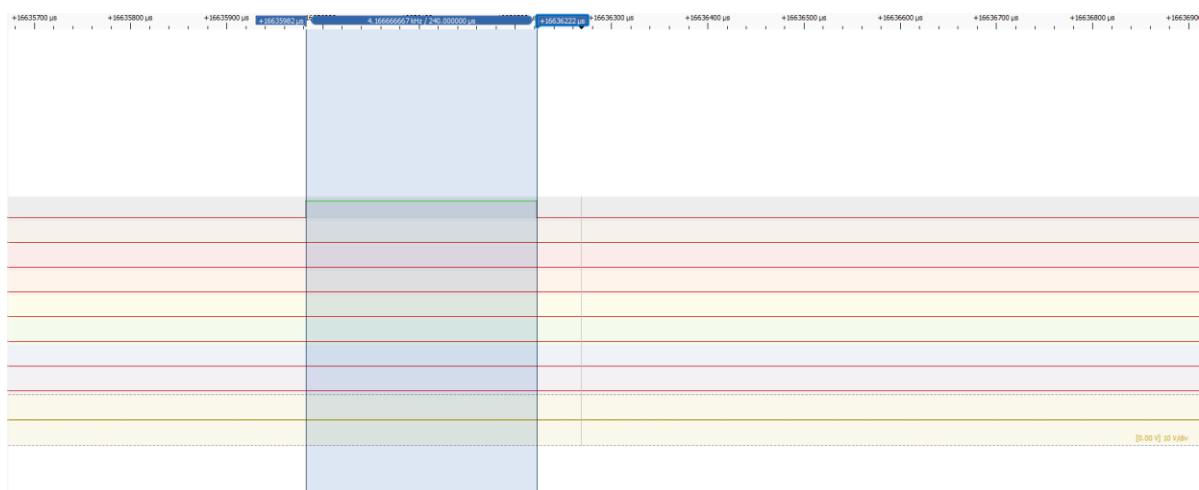
Como puede comprobarse gracias a la captura del pulso con el analizador lógico y el uso de Pulseview, la emisión de este pulso es correcta.

2º) Sensor lee y calcula correctamente la distancia hacia un objeto:

Con objeto a comprobar que el sensor está leyendo correctamente la distancia suya a cualquier objeto que tenga dentro del rango de alcance, se diseña esta prueba. Tanto con el ‘Pulseview’ como con la ventana ‘Serial Printf’ se va a leer la distancia desde este sensor al objeto.

Condiciones de entrada:

- Inicialización correcta del hilo.
- Inicialización correcta de ambos pines: PB2 y PB6
- Conexión correcta del sensor a los pines de la STM32F429ZI
- Lectura correcta de la distancia
- Conversión correcta de la distancia



Como puede comprobarse, se ha capturado el pulso leído del sensor, que dividiendo ese ancho entre 58 para pasar el tiempo a cm, se obtienen unos 4,1 cm. En su momento no se hizo captura de la ventana ‘Debug Serial Printf’, pero se da como válida y superada a esta prueba.

3º) Compruebo que el sensor permite leer el agua en un vaso:

El objetivo de esta prueba es conseguir leer correctamente el nivel de agua de un vaso colocando el sensor en la parte de arriba de este y comprobar dicha capacidad en la ventana ‘Debug Serial Printf’. Dicha capacidad ha de obtenerse entre un porcentaje de 0-100%. Para ello, el módulo es configurable para adaptar al código a la capacidad del depósito.

Condiciones de entrada:

- Inicialización correcta del hilo.
- Inicialización correcta de ambos pines: PB2 y PB6
- Conexión correcta del sensor a los pines de la STM32F429ZI
- Lectura correcta de la distancia
- Conversión correcta a capacidad entre un porcentaje de 0% a 100%.

Esta prueba se dio como exitosa, ya que se consiguió leer de forma correcta el porcentaje del depósito de agua, además variaba el valor del porcentaje si se añadía o se retiraba agua.

4º) Dependiendo de la cantidad de agua del depósito enciende un color del RGB:

Para terminar este módulo se decide testear este último caso y es que dependiendo del porcentaje de nivel de agua del depósito, se encienda un color del led RGB de la tarjeta mbed de aplicaciones.

Condiciones de entrada:

- Inicialización correcta del hilo.
- Inicialización correcta de ambos pines: PB2 y PB6
- Conexión correcta del sensor a los pines de la STM32F429ZI
- Lectura y conversión correcta de la capacidad del depósito
- Inicialización correcta de los pines que inicializan al led RGB: PD11, PD12 y PD13
- Conexión correcta de la tarjeta mbed de aplicaciones

El resultado de esta prueba es exitoso, ya que dependiendo del porcentaje de agua del depósito se encendía un color distinto: 0%-20%, color rojo; 20%-70%, color naranja; 70%-90%, color verde.

SENSOR BH1750

El objetivo principal de la prueba realizada durante la validación de este módulo fue comprobar el correcto funcionamiento del sensor de luminosidad BH1750. Para ello, se requería asegurar el funcionamiento adecuado de varios elementos clave del sistema: la correcta inicialización del bus I²C correspondiente, la adecuada inicialización y configuración del sensor de luminosidad BH1750 y la verificación de integridad tanto en la transmisión como en la recepción de datos por el bus, verificando de los valores obtenidos son coherentes.

Para verificar esto último, se optó por realizar una prueba funcional básica basada en la creación de una variable global de tipo float, denominada lum, asociada al módulo del sensor. Esta variable fue inicialmente inicializada a 0.0 y posteriormente actualizada de forma periódica (cada 1 segundo) con el valor devuelto por la función Brightness_Reading (), encargada de obtener y convertir la medida de luminosidad proporcionada por el sensor en unidades de lux basándose en las especificaciones indicadas por el fabricante.

Condiciones de entrada:

- Correcta inicialización del hilo.
- Correcta inicialización del bus I²C.
- Correcta inicialización del sensor BH1750.
- Coherencia en los valores de luminosidad devueltos por el sensor

Durante la fase de pruebas, la variable lum fue monitorizada a través de la ventana "Watches" del entorno de desarrollo Keil uVision, permitiendo así observar en tiempo real la evolución de los valores medidos por el sensor bajo distintas condiciones de iluminación. Este procedimiento permitió confirmar que el sensor respondía correctamente ante variaciones en la luz ambiente, validando así la funcionalidad general del módulo.

COMUNICACIÓN ENTRE PLACAS – UART

Para probar el bloque de la comunicación entre placas se han creado dos hilos para cada una de las placas (MÁSTER y SLAVE). Un hilo se encarga de la transmisión de mensajes y otro de la recepción. Se habilitó el led 2 (AZUL) de la placa STM32, para comprobar visualmente el envío de los mensajes.

CONDICIONES DE ENTRADA:

Configuración de hardware:

- Dos placas STM32 conectadas entre sí mediante UART
- Alimentación y conexión física correcta de TX y RX entre ambas placas (Masas de las placas conectadas entre sí).
- LED azul (LED 2) habilitado en ambas placas para indicar actividad de transmisión/recepción.

Configuración de software:

- Sistema operativo en tiempo real (CMSIS RTOS2) activo en ambas placas.
- Dos hilos definidos por placa:
 - Uno para transmisión.
 - Otro para recepción y validación de la trama.
- Función generadora de datos simulados de sensores en el SLAVE.
- Implementación del protocolo de trama con:
 - Byte de inicio: 0x7E
 - Datos del sensor: valores aleatorios formateados
 - Byte de fin: 0x7F

Condiciones de ejecución:

- En el modo **periódico**, el SLAVE envía tramas automáticamente cada segundo.
- En el modo **bajo solicitud**, el MÁSTER envía una trama [0x7E, 0x01, 0x7F], la cual el SLAVE reconoce y responde con una trama de datos.
- Comunicación UART habilitada y configurada a una velocidad de baudios adecuada (9600 bps).

Condiciones de observación/verificación:

- Debug activo con puntos de ruptura en las variables que almacenan la recepción de tramas.
- Visualización de las tramas y datos en el Debug Viewer.
- Observación del parpadeo del LED azul como indicación visual de transmisión y recepción.

Comunicación SLAVE-MÁSTER

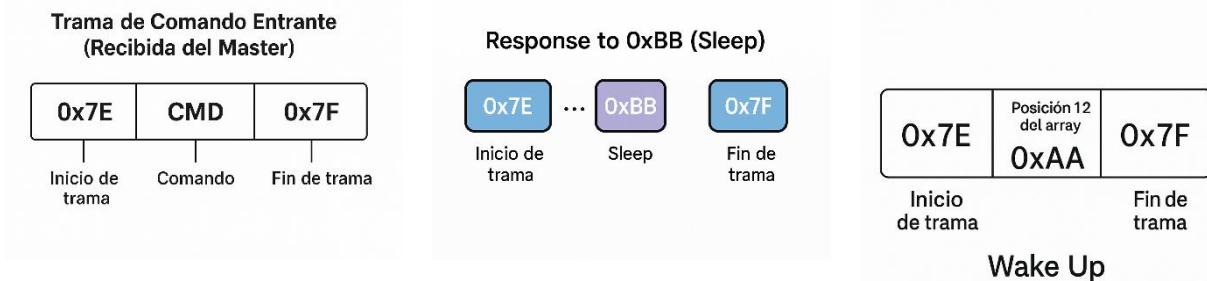
Este led parpadea cada vez que el SLAVE envía las medidas y el MÁSTER las recibe, cada uno con su respectivo led de su placa. Para simular los datos que se envían, se creó una función que genera valores aleatorios correspondientes con los datos de los sensores conectados en la placa SLAVE. Se forma la trama con estos valores, un byte de inicio 0x7E y un byte de fin 0x7F. Las tramas se envían al MÁSTER cada segundo y luego se ha hecho una prueba, enviando las tramas cuando el MÁSTER las solicite mediante una trama.

Comunicación MÁSTER-SLAVE

Se crea una trama [0x7E, 0x01, 0x7F] y se envía al SLAVE. Este se encarga de recibirla y enviar las medidas al MÁSTER.

Para comprobar el correcto funcionamiento de la comunicación, se ha abierto en el watch la variable donde se reciben los datos en cada una de las placas (puntos de ruptura), en el debug viewer se imprimen los valores convertidos a las unidades correspondientes de cada medida y, por último, en el pulseView se observan las tramas byte a byte que se envían entre el MÁSTER y SLAVE.

TRAMA ENVIADA POR EL MÁSTER AL SLAVE TRAMA DE RESPUESTA DEL SLAVE AL CMD 0xBB TRAMA DE RESPUESTA DEL SALVE AL CMD 0xAA



El resultado de la prueba ha sido satisfactorio. Las tramas generadas por el SLAVE, compuestas por un byte de inicio (0x7E), datos simulados correspondientes a sensores, y un byte de fin (0x7F), fueron correctamente recibidas por el MÁSTER. Además, se verificó la comunicación bidireccional con éxito, permitiendo al MÁSTER enviar tramas de solicitud específicas ([0x7E, 0x01, 0x7F]) y recibiendo en respuesta los datos del SLAVE.

PRUEBAS CON TODOS LOS MÓDULOS - UART

Una vez probado el funcionamiento de todos los módulos por separado es el momento de juntarlos todos en el mismo programa. En la placa SLAVE añadimos los módulos de los sensores BME680, HC-SR04 y BH1750. Estos hilos se comunican, mediante colas con el hilo, que se encarga de la transmisión de la comunicación entre placas, además se ha creado un autómata con las diferentes tramas que puede recibir del MÁSTER. En la placa MÁSTER, se han incluido los módulos del RFID y la memoria AT24C256. Al igual, que en el SLAVE se ha diseñado un autómata para determinar, donde nos encontramos (PRESENCIAL, STANBY y WEB).

Condiciones de hardware

- SLAVE:
 - Conexión y alimentación correcta de los siguientes sensores:
 - BME680 (temperatura, humedad, gas)
 - HC-SR04 (distancia ultrasónica)
 - BH1750 (iluminación)
 - Comunicación UART activa con la placa MÁSTER.
 - LED de estado (por ejemplo, LED2 azul) operativo para indicación visual.
- MÁSTER:
 - Conexión estable de:
 - Módulo RFID (lectura de tarjetas)
 - Memoria EEPROM AT24C256 (almacenamiento persistente)
 - UART operativa para comunicarse con el SLAVE.

Condiciones de software

- SLAVE:
 - RTOS habilitado con hilos independientes para:
 - Adquisición de datos de cada sensor (BME680, HC-SR04, BH1750).
 - Hilo de comunicación UART.
 - Colas RTOS activas para transmitir datos desde los hilos sensores al hilo de comunicación.
 - Autómata de recepción UART implementado para procesar tramas entrantes del MÁSTER y responder según el tipo de instrucción (por ejemplo, envío de medidas completas, solo distancia, etc.).
- MÁSTER:
 - Hilos de lectura de RFID y de acceso a la memoria EEPROM correctamente inicializados.
 - Autómata de estados del sistema operativo con las siguientes fases:
 - PRESENCIAL: interacción directa con usuario o sensores.
 - STANDBY: espera inactiva o ahorro energético.
 - WEB: modo de operación remota o carga a servidor.
 - Lógica definida para envío de tramas específicas al SLAVE según el estado del sistema.

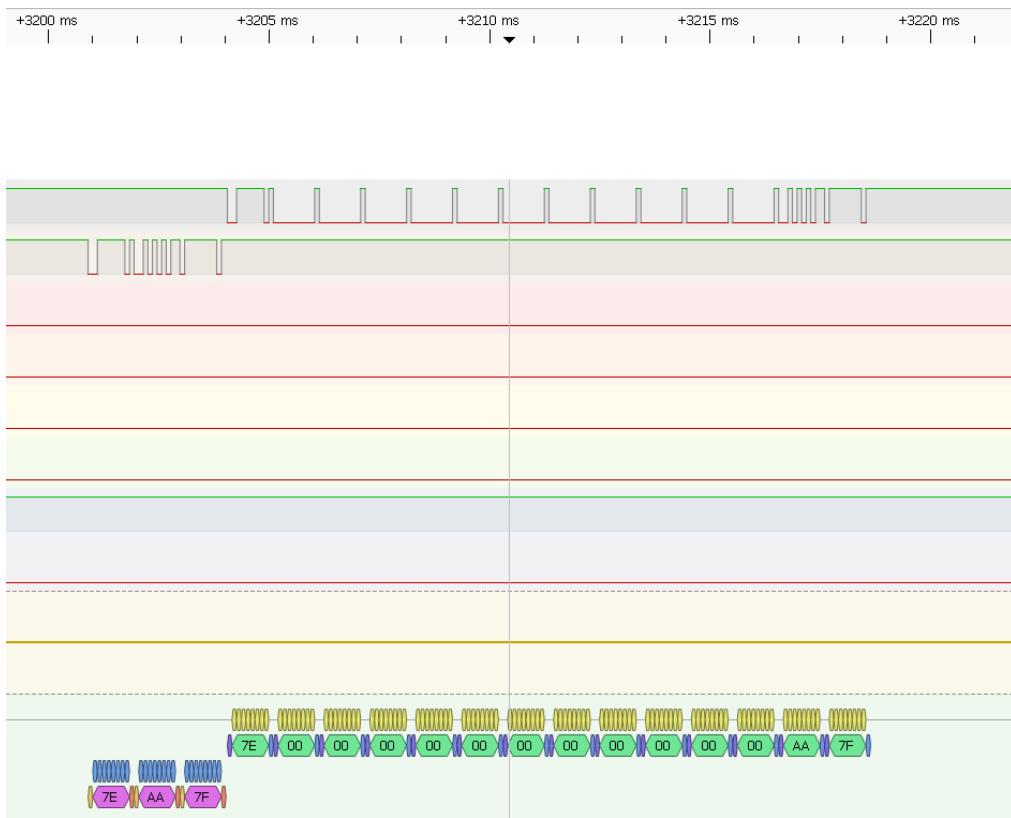
Condiciones de ejecución:

- Ambas placas están encendidas y sincronizadas.
- UART bidireccional funcional.
- Al menos una tarjeta RFID registrada previamente en EEPROM.
- El sistema se encuentra en un estado válido del autómata (ej. comienza en STANDBY).
- Se inician eventos que fuerzan el paso entre estados del autómata (ej. detección de tarjeta RFID, tiempo de espera, orden remota).
- Las tramas recibidas del SLAVE deben contener datos coherentes según el sensor solicitado por el MÁSTER.

Para comprobar el correcto funcionamiento de la comunicación de las placas y que llegan los valores correctos de los sensores, se ha utilizado el analizador de espectros para visualizar en el programa PulseView, la comunicación entre las placas. Para comprobar la integridad de la comunicación, se visualizaba en el Debug Viewer las medidas que recibía el MÁSTER, y se verificaban con las emitidas por el SLAVE. A continuación, se podrán observar las diferentes tramas que se han capturado con el analizador.

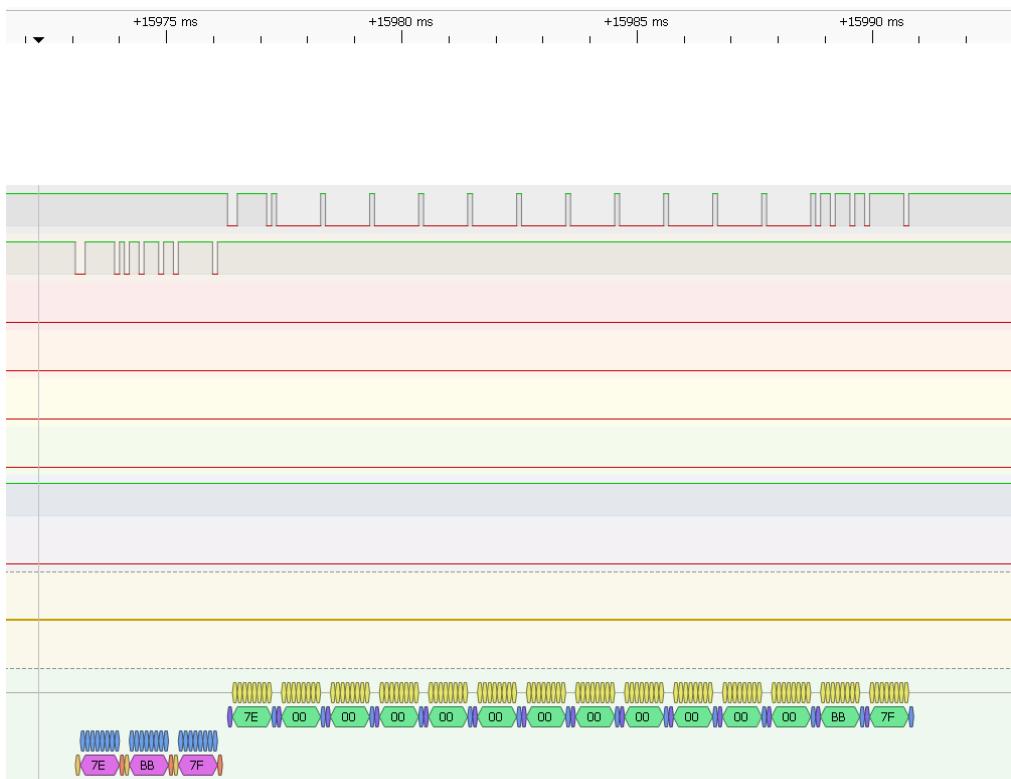
- Trama dormir al SLAVE con ACK de confirmación

En la imagen se puede observar que el MÁSTER envía la trama con contenido “AA” para dormir al SLAVE, y le responde con un ACK de confirmación enviando el mismo comando “AA” en el byte número 13.

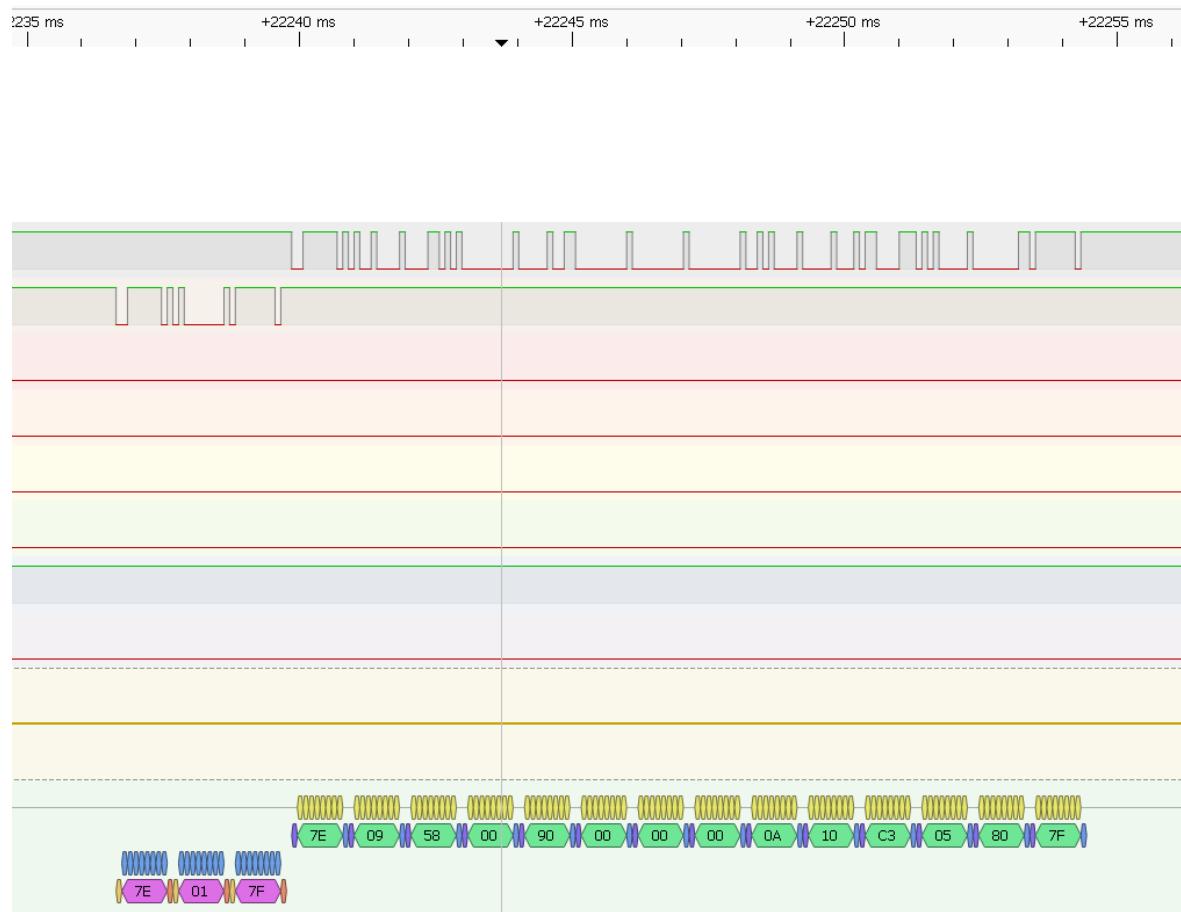


- Trama despertar al SLAVE con ACK de confirmación

En la imagen se puede observar que el MÁSTER envía la trama con contenido “BB” para despertar al SLAVE, y le responde con un ACK de confirmación enviando el mismo comando “BB” en el byte número 13.



- Trama pedir medidas más la respuesta del SLAVE con todas las medidas
 La primera trama captada en la secuencia corresponde a una transmisión iniciada por el MÁSTER. Esta tiene como finalidad solicitar al SLAVE la adquisición y transmisión de un conjunto de medidas de todos los sensores, mediante el comando 0x01, que se ha incluido en la trama.
 La segunda trama representa la respuesta del SLAVE, quien, tras procesar la solicitud anterior, devuelve un conjunto de datos correspondientes a las medidas solicitadas. Esta información está compuesta por varios bytes, cada uno representando una parte de las medidas adquiridas anteriormente.



Tras el análisis de la imagen capturada, se concluye que todas las tramas de comunicación entre MÁSTER y el dispositivo SLAVE han sido correctamente transmitidas y recibidas, sin errores de sincronización, direccionamiento ni integridad de datos. La estructura de ambas tramas —la de solicitud emitida por el MÁSTER y la de respuesta generada por el SLAVE— se ajusta a la secuencia esperada según el protocolo implementado.

Asimismo, se ha verificado que los datos recibidos en el MÁSTER coinciden fielmente con las medidas esperadas, tanto en contenido como en formato. Esto confirma que el proceso de adquisición, transmisión y decodificación de las medidas se ha realizado de forma íntegra y fiable, validando así el correcto funcionamiento del sistema de comunicación.

FUNCIONAMIENTO DE LA PÁGINA WEB

Con el objetivo de verificar el correcto funcionamiento de la página web integrada en el proyecto, se llevó a cabo un proceso de depuración progresiva y manual, centrado en validar cada uno de los apartados de la interfaz con los que cuenta.

Debido a que es imposible realizar una depuración paso a paso sobre el código web directamente desde el entorno de desarrollo Keil uVision como venimos estando acostumbrados, la única forma de poder ir comprobando si el funcionamiento de esta es el esperado o no es ir cargando el nuevo código en la placa tras cada cambio en los archivos relacionados con la web (como .cgi, .htm o scripts .js), y acceder a esta para así ir comprobando que los datos mostrados, botones y elementos interactivos funcionaban tal y como se esperaba. Para ello, fue necesario realizar múltiples ciclos de edición, compilación, carga y verificación.

Gracias a esta depuración iterativa, fue posible ajustar el comportamiento y el diseño de cada sección de la web hasta alcanzar el funcionamiento estable y representativo deseado.

FUNCIONAMIENTO DEL BAJO CONSUMO

Con el objetivo de comprobar el correcto funcionamiento del modo sleep en la placa esclava (SLAVE), se ha implementado un mecanismo de activación por evento externo, utilizando para ello el pin PB1.

En el MÁSTER, el pin PB1 ha sido configurado como salida GPIO, permitiendo generar flancos ascendentes (de 0 a 1 lógico) de forma controlada. Estos flancos actúan como señal de activación externa hacia el esclavo.

Por su parte, en la placa SLAVE, el pin PB1 ha sido configurado como entrada con detección de interrupción por flanco de subida mediante la siguiente rutina de inicialización:

```
void Init_PB1_WAKEUP (void){  
    GPIO_InitTypeDef GPIO_InitStruct;  
  
    __HAL_RCC_GPIOB_CLK_ENABLE(); // Se habilita el reloj del GPIO C dado que el botón está asociado al GPIOC13  
  
    GPIO_InitStruct.Pin = GPIO_PIN_1;  
    GPIO_InitStruct.Pull = GPIO_PULLDOWN; // No hay R pullup ni pulldown  
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING; // Detecta los flancos de bajada  
  
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct); // Se inicia el pin  
  
    /* Enable and set Button EXTI Interrupt to the lowest priority */  
    HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0x0F, 0x00); /* The lower is the priority, the more prioritary is*/  
    HAL_NVIC_EnableIRQ(EXTI15_10_IRQn); // Las interrupciones del botón azul corresponden a las EXTI15-10  
}
```

Mediante esta configuración, el sistema permite que el SLAVE, estando en modo sleep, despierte automáticamente al detectar un flanco ascendente en el pin PB1. Esta interrupción externa actúa como mecanismo de wake-up, restableciendo la ejecución del código desde una condición de bajo consumo.

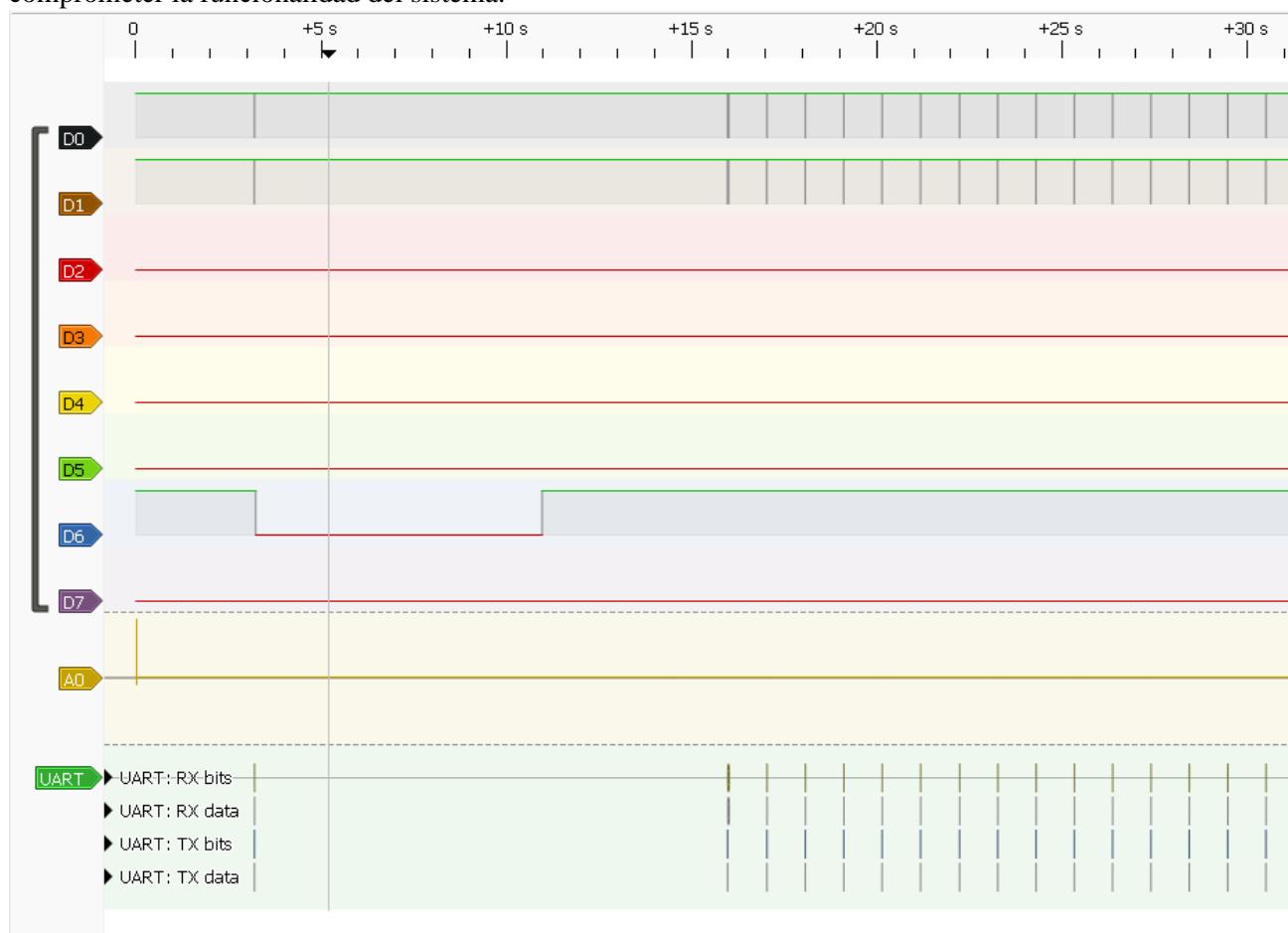
```
/* Private function prototypes -----*/  
void EXTI15_10_IRQHandler (void);  
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_PIN);  
/* Private functions -----*/  
void EXTI15_10_IRQHandler (void){  
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);  
}  
  
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_PIN){  
}
```

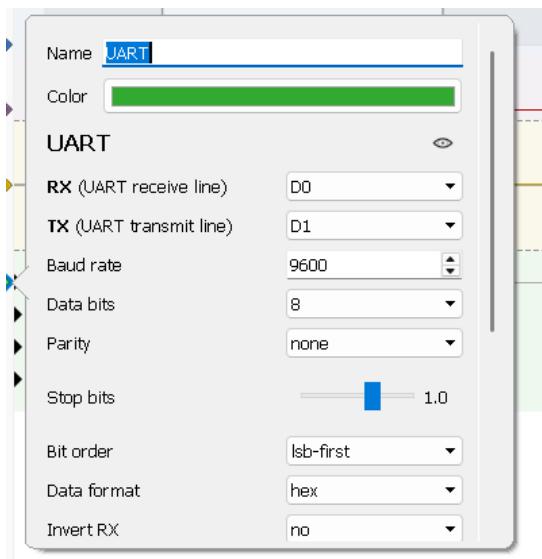
Para inducir los flancos ascendentes necesarios en el pin **PB1**, el MÁSTER establece dicho pin a nivel lógico alto ('1') en tres situaciones específicas:

- **Cuando se detecta una tarjeta RFID,**
- **Cuando se pulsa un botón en la interfaz web,**
- **Cuando se produce una alarma del sistema.**

En todos estos casos, la transición de bajo a alto en el pin PB1 genera un flanco de subida, el cual es detectado por el SLAVE como una interrupción externa que permite salir del modo *sleep*. Una vez que la condición que motivó el evento (lectura RFID, interacción web o alarma) finaliza, el maestro restablece el nivel del pin PB1 a estado bajo ('0'), lo que permite que el SLAVE vuelva a entrar en modo de bajo consumo tras completar la ejecución de las funciones asociadas a la interrupción (por ejemplo, las contenidas en la callback de la alarma).

Se ha verificado satisfactoriamente el correcto funcionamiento del sistema de bajo consumo implementado en la placa SLAVE. Mediante el control del pin PB1 desde el MÁSTER, se ha logrado generar eventos externos que provocan flancos ascendentes capaces de despertar al SLAVE desde el modo *sleep*. Posteriormente, tras completar las tareas requeridas por cada evento (tales como gestión de alarmas o lectura de identificadores), el sistema retorna automáticamente al estado de bajo consumo. Esta validación confirma que el mecanismo de entrada/salida del modo *sleep* funciona de forma eficiente, permitiendo optimizar el consumo energético sin comprometer la funcionalidad del sistema.





Una señal clave en esta traza es la correspondiente al pin **PB1**, asociado a la línea **D6**. Esta línea se utiliza como interrupción externa para despertar al SLAVE del modo de bajo consumo. Durante la primera parte del registro, la señal D6 se mantiene en nivel bajo, indicando que el SLAVE permanece en estado "sleep". En un instante determinado, el MÁSTER establece esta línea en nivel alto, generando un flanco de subida. Esta transición es detectada por el SLAVE como una interrupción externa, lo que provoca su activación inmediata. Una vez activo, el SLAVE ejecuta las tareas correspondientes al evento que originó la activación. Estas pueden ser, por ejemplo, la lectura de una tarjeta RFID, una orden enviada desde la interfaz web, o el tratamiento de una alarma. Mientras el SLAVE se encuentra activo, también se observa actividad en las líneas UART, lo que confirma que se están realizando comunicaciones entre el MÁSTER y el SLAVE durante ese periodo.

Finalmente, cuando el evento ha sido atendido y completado, el MÁSTER devuelve la señal PB1 (D6) a nivel bajo. Esta acción permite que el SLAVE vuelva a entrar en modo de bajo consumo. La correcta secuencia de activación, operación y retorno al estado "sleep" se repite según se requiera, garantizando un uso eficiente de la energía del sistema.

Este análisis confirma el buen funcionamiento del sistema de interrupciones y gestión de energía. El control del pin PB1 desde el MÁSTER permite inducir eventos que despiertan al SLAVE sólo cuando es necesario, evitando consumos innecesarios durante los períodos de inactividad. Además, la actividad en la UART durante los ciclos de activación refuerza que el SLAVE responde correctamente a cada interrupción, cumpliendo con sus tareas y optimizando el consumo energético sin afectar la funcionalidad general del sistema.

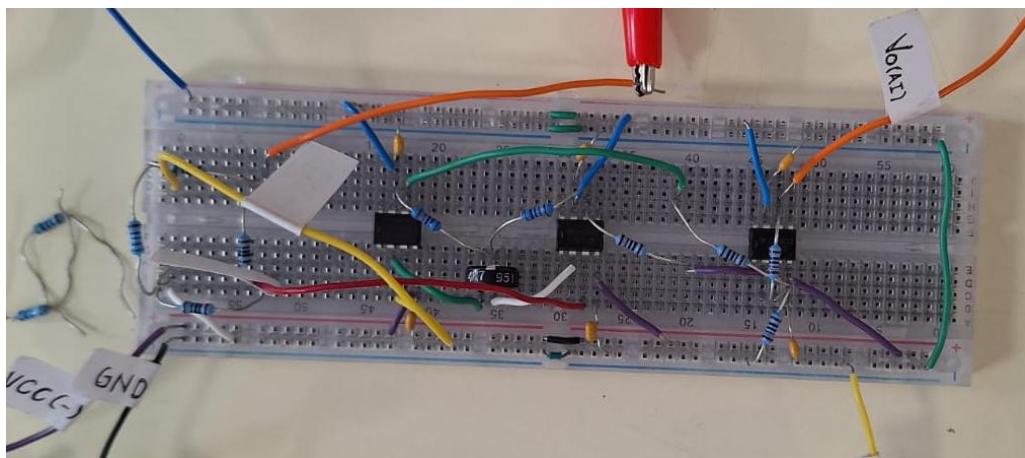
5.2 Pruebas realizadas hardware.

Descripción del método de prueba utilizado para comprobar que cada subsistema funciona de acuerdo a las especificaciones formuladas.

5.2.1 Acondicionador de temperatura

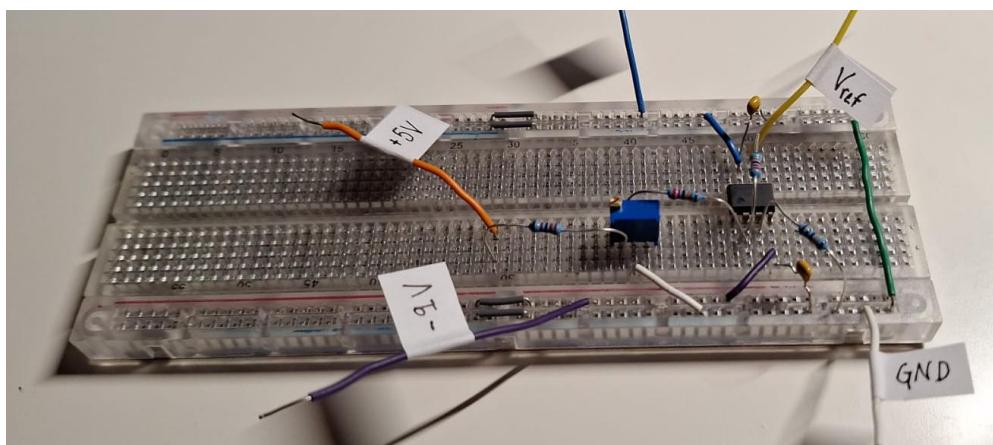
1. Construcción del amplificador de instrumentación

Lo primero que se desarrolló del acondicionador de temperatura fue todo lo relacionado al AI en sí. Para verificar el correcto funcionamiento del circuito, las pruebas que se realizaron consistieron en ir simulando una serie de temperaturas mediante su resistencia equivalente y se fue comprobando si la tensión obtenida a la salida coincidía con la esperada. En dichas pruebas se observó como las tensiones obtenidas contaban con un offset, por lo que se decidió implementar un amplificador seguidor con la finalidad de corregirlo.



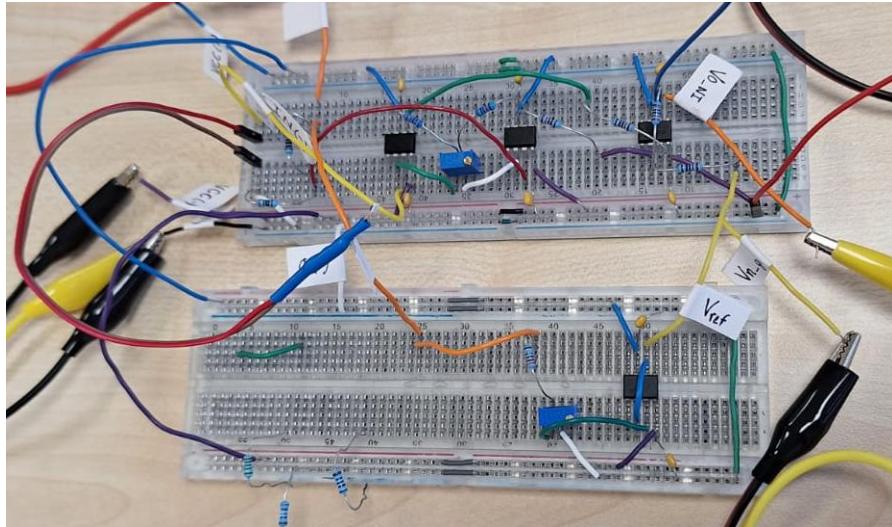
2. Ajuste de offset para el amplificador de instrumentación

Tras observar que las tensiones obtenidas a la salida del AI contaban con un offset, se decidió implementar un AO con topología seguidor junto con un divisor de tensión con el fin de ser capaces de corregir así el offset observado. Para ello, se implementó en otra protoboard independiente el circuito correspondiente, y se verificó que el mismo proporcionaba un rango de tensiones dentro del cual se encontraba los valores requeridos para corregir el offset del AI.



3. Integración completa del acondicionador RTD

Tras realizar las respectivas comprobaciones de cada uno de los dos circuitos diseñados y verificar que ambos funcionaban correctamente, se terminó juntando ambas protoboard, realizando las conexiones oportunas y se verificó que, simulando la misma temperatura con su resistencia equivalente, la tensión a la salida iba variando cuando se modificaba el valor del potenciómetro el AO con topología de seguidor.



4. Ajuste del acondicionador en la PCB

Una vez comprobado el correcto funcionamiento de todo el circuito, se procedió a realizar el ajuste y caracterización de este dentro del rango de valores de tensión que puede soportar el pin ADC de la STM32, el cuál es de 0 a 3,3V. Dicho ajuste y caracterización se realizó empleando las resistencias mas próximas a los extremos con las que contamos. Dichos valores resistivos serán de 99,688 y 120,01 Ω , cuyas temperaturas equivalentes son -0,85 y 51,95 °C.

Tras esto, procederemos a realizar la corrección del offset. Para ello, teóricamente utilizaremos la resistencia equivalente a los -5 °C que corresponde a los 0 V y calcularemos el valor de n de la recta, que corresponderá con la tensión de offset.

Finalmente, realizaremos la caracterización del acondicionador empleando todas las resistencias con las que contamos que se encuentran dentro del rango de valores resistivos para los que ha sido ajustado el circuito.

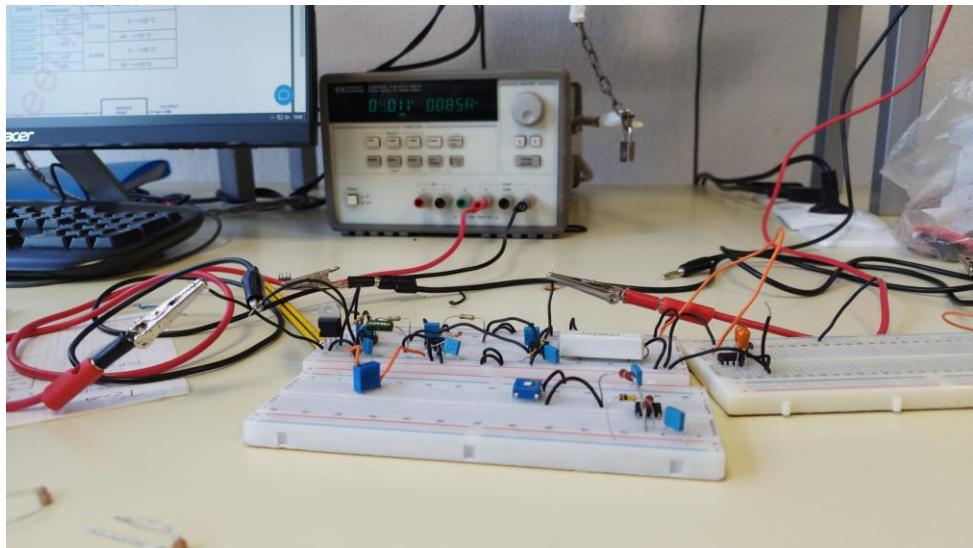
5.2.2 Acondicionador para el consumo

1. Montaje y prueba de circuito

Antes de mandar a fabricar la PCB final, montamos el circuito de alimentación y el acondicionador de consumo sobre una placa prototipo para hacer las primeras pruebas. Como todavía no teníamos las baterías disponibles, alimentamos todo el sistema con una fuente de tensión de laboratorio ajustable.

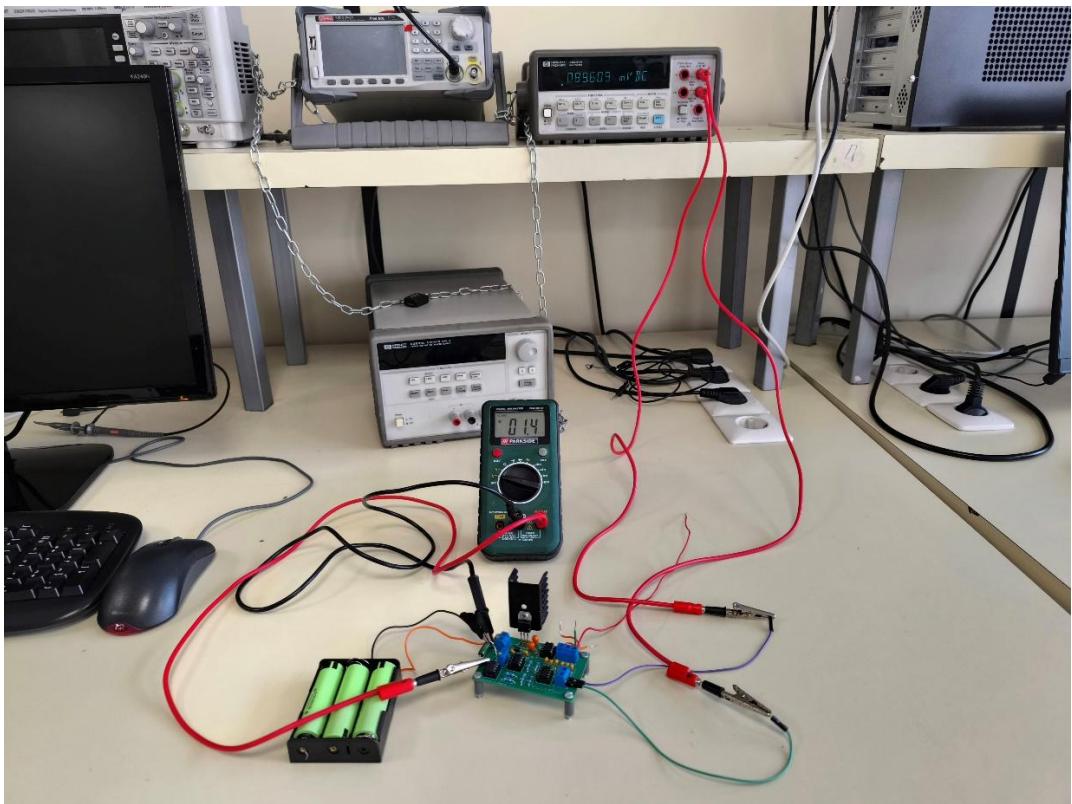
Una de las comprobaciones más importantes en esta fase fue asegurarnos de que el circuito generaba correctamente la tensión negativa necesaria para el acondicionador del sensor de consumo. Para conseguir esos -9V a partir de una entrada de 9V, usamos el integrado LMC7660, y durante la prueba vimos que el circuito funcionaba como esperábamos: obteníamos la tensión negativa sin problemas. Esto nos confirmó que tanto el LMC7660 como los componentes que lo rodean (condensadores, conexiones, pines, etc.) estaban bien montados.

También aprovechamos para revisar que el cableado estaba bien hecho, que las masas y las tensiones se distribuían correctamente por todo el sistema, y que no había conflictos entre los distintos módulos electrónicos. Esta prueba fue clave para asegurarnos de que el diseño del esquemático estaba correcto antes de mandar a fabricar la placa definitiva.



2. Ajuste de la ganancia

Se realizó un ajuste preciso del potenciómetro de ganancia (RG) para alcanzar una ganancia teórica objetivo de aproximadamente 64. Se utilizó un potenciómetro de 10 kΩ inicialmente, posteriormente reemplazado por uno de 5 kΩ tipo trimmer multivuelta (modelo 3296Y-5K) para mejorar la estabilidad y minimizar la variabilidad de salida.



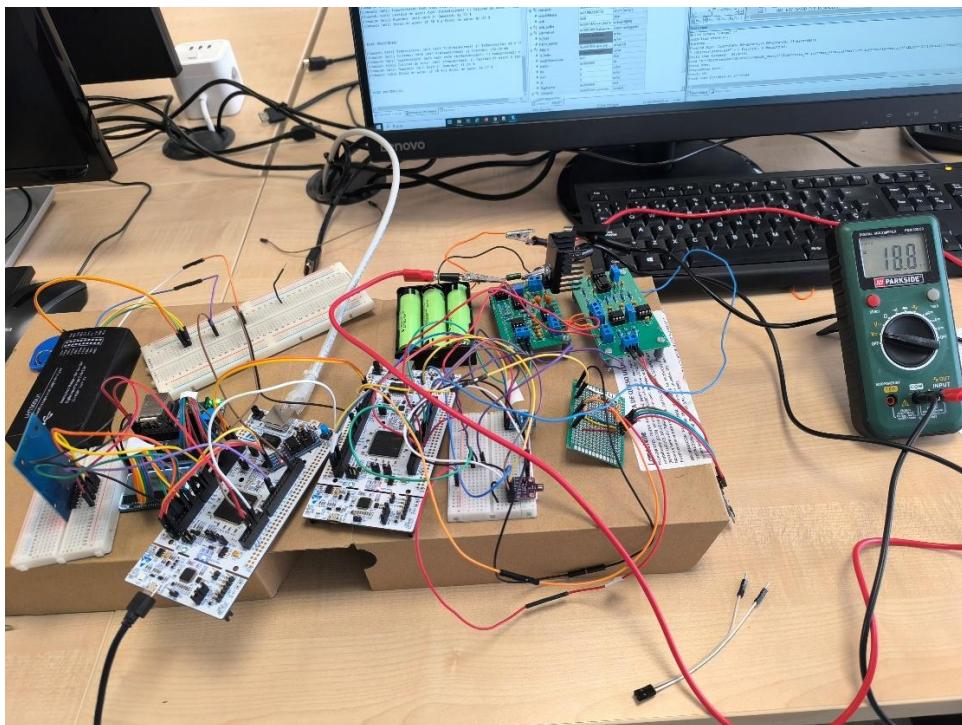
En la imagen adjunta puede observarse el ajuste de ganancia. El multímetro inferior está conectado a las bornas de la resistencia shunt, mientras que el multímetro superior mide la salida del acondicionador de señal. La relación entre ambas tensiones confirma el valor de ganancia esperado ($G \approx 64$), de acuerdo con la fórmula del amplificador de instrumentación:

$$G = 1 + \frac{2R}{R_g}$$

donde $R=100 \text{ k}\Omega$ y R_g fue ajustado hasta aproximadamente $3 \text{ k}\Omega$.

3. Verificación del consumo total del sistema

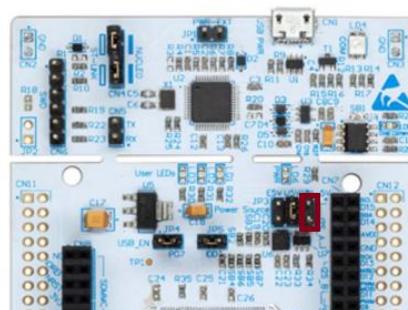
Con el fin de comprobar el correcto funcionamiento conjunto de todos los subsistemas integrados (acondicionador de señal para RTD, acondicionador de consumo con resistencia shunt y el microcontrolador STM32 slave), se realizó una medición del consumo total del sistema.



El consumo total del sistema fue medido con un multímetro conectado en serie a la alimentación principal (en la resistencia shunt), mostrando un valor de aproximadamente 188mA. Esta medición verifica que todos los módulos se encuentran activos y funcionando correctamente de forma simultánea.

4. Conexión de la alimentación autónoma a la tarjeta slave

Para poder usar la alimentación autónoma en la tarjeta slave es necesario hacer uso del jumper JP3 en la posición VIN-5V. El procedimiento es cargar el proyecto con el botón LOAD del Keil teniendo el jumper en medio. Una vez hecho esto, debemos de quitar el USB, mover el JP3 a la posición VIN-5V y conectar las baterías.



Después, de nuestro circuito de alimentación sacamos el cable que vaya a alimentar a la STM y lo conectamos al pin VIN junto a GND en el banco de conectores CN8:



6 CONCLUSIONES.

Este proyecto ha abordado el diseño, desarrollo e integración de un sistema inteligente orientado al control y monitorización de un invernadero, combinando electrónica analógica, digital y software embebido. A través de una arquitectura distribuida tipo MÁSTER-SLAVE, se ha logrado implementar una solución robusta que permite medir variables ambientales clave (temperatura, humedad, luminosidad, calidad del aire y nivel de agua), visualizar la información mediante una interfaz web y controlar el consumo energético gracias a mecanismos de bajo consumo y activación selectiva.

Se ha validado con éxito la comunicación UART entre placas, la funcionalidad de sensores conectados por protocolos I²C y GPIO, así como el correcto almacenamiento de medidas en una memoria EEPROM externa. Uno de los aspectos más destacables ha sido la integración eficiente del sistema de gestión de energía, que permite al SLAVE operar en modo "sleep" y activarse únicamente ante eventos relevantes, lo cual optimiza notablemente el consumo.

Además, se ha desarrollado un sistema de acondicionamiento de señal para la lectura de corriente a través de una resistencia shunt, con un amplificador instrumental calibrado con precisión. Esto ha permitido obtener medidas fiables y realizar un análisis de consumo de cada bloque funcional. Todas las fases del diseño (alimentación, medida, control y visualización) han sido documentadas, validadas experimentalmente y contrastadas con cálculos teóricos, lo que otorga solidez al desarrollo.

En definitiva, se ha conseguido construir un sistema funcional, escalable y energéticamente eficiente, capaz de adaptarse a escenarios reales de automatización y monitorización ambiental, cumpliendo así con los objetivos establecidos al inicio del trabajo.

7 PRESUPUESTO FINAL.

El presupuesto final del proyecto es el siguiente, siendo **91,4€ el total** y tocando a **22,85€ por estudiante**:

Componente	Unidades	Precio unitario (€)	Subtotal (€)
Sistemas embebidos			0,00 €
Tarjeta núcleo STM32F429ZI (*)	2	0,00 €	0,00 €
Tarjeta mbed de aplicaciones (*)	1	0,00 €	0,00 €
Sensores integrados			6,81 €
Memoria EEPROM AT24C256	1	1,01 €	1,01 €
LECTOR RFID RC522 + Tarjetas (*)	1	0,00 €	0,00 €
BME680	1	5,80 €	5,80 €
BH1750 (*)	1	0,00 €	0,00 €
HC-SR04 (*)	1	0,00 €	0,00 €
Placas de Circuitos Impresos (PCBs)			14,42 €
Alimentación + Acondicionador consumo	5	0,35 €	1,75 €
Acondicionador RTD	5	0,68 €	3,40 €
Tasas, envío, etc.	1	9,27 €	9,27 €
Componentes: Alimentación + Consumo			29,57 €
Pilas 18650 NiMH 3500 mAh 3.7V	8	1,94 €	15,49 €
Carcasa pilas	1	2,80 €	2,80 €
Condesadores de acople (100 nF)	8	0,12 €	0,96 €
Condensadores polarizados (10 µF)	2	0,05 €	0,10 €
Diodo de protección (1N4007)	1	0,03 €	0,03 €
Regulador de tensión: LM7809	1	1,57 €	1,57 €
Convertidor de tensión: LMC7660	1	1,56 €	1,56 €
Resistencias AI (10 kΩ)	6	0,02 €	0,12 €
Rshunt (0,1 Ω)	1	0,30 €	0,30 €
Potenciómetro (10 kΩ)	1	0,70 €	0,70 €
Amplificadores operacionales (OP07)	3	1,60 €	4,80 €
Conectores, jumpers, etcétera	3	0,38 €	1,14 €
Componentes: Acondicionador RTD			26,20 €
Condesadores de acople (100 nF)	8	0,12 €	0,96 €
RTDs	2	7,65 €	15,30 €
Amplificadores operacionales (OP07)	4	1,60 €	6,40 €
Resistencias AI (10 kΩ)	7	0,10 €	0,70 €
Resistencias Puente Wheatstone	3	0,10 €	0,30 €
Potenciómetro (20 kΩ)	1	0,70 €	0,70 €
Potenciómetro (1 kΩ)	1	0,70 €	0,70 €
Conectores, jumpers, etcétera	3	0,38 €	1,14 €
Miscelánea			14,40 €
Tornillos 3 mm PCBs	8	0,10 €	0,80 €
Cables macho-macho	1	8,00 €	8,00 €
Liston de madera(2,40m)	3	1,50 €	4,50 €
Plantas falsas	1	1,10 €	1,10 €

(*) Los estudiantes ya poseían estos medios antes de comenzar el proyecto

8 EQUIPO DE TRABAJO.

Miembro	Tareas iniciales	Tareas desarrolladas	Tiempo dedicado
Aitor	Circuito acondicionador RTD	Circuito acondicionador RTD	25 horas
	Implementación sensor RFID RC522	Implementación sensor RFID RC522	8 horas
	Sensor HC-SR04 + RGB	Sensor HC-SR04 + RGB	4 horas
	Módulo zumbador	Módulo zumbador	1 hora
	No asignado	Diseño módulo LCD + Nacho	2 horas
	Servidor web	Servidor web	10 horas
	Desarrollo de la memoria	Desarrollo de la memoria	20 horas
	No asignado	Bajo consumo + Nizar	8 horas
Álvaro	Circuito acondicionador RTD	Circuito acondicionador RTD	25 horas
	Sensor BH1750	Sensor BH1750	10 horas
	Módulo RTC + SNTP	Módulo RTC + SNTP	10 horas
	Módulo ADCs	Módulo ADCs	5 horas
	Servidor web	Servidor web	40 horas
	No asignado	Diseño módulo LCD + Nacho	1 hora
	No asignado	Bajo consumo + Nizar	5 horas
	Desarrollo de la memoria	Desarrollo de la memoria	10 horas
Nachó	Circuito consumo + alimentación	Circuito consumo + alimentación	35 horas
	Sensor BME680	Sensor BME680	25 horas
	Comunicación entre tarjetas núcleo	Comunicación entre tarjetas núcleo	10 horas
	Principal Máster	Principal Máster	15 horas
	Principal Slave	Principal Slave	5 horas
	Diseño módulo LCD	Diseño módulo LCD + Aitor + Álvaro	1 hora
	Desarrollo de la memoria	Desarrollo de la memoria	15 horas
Nizar	Circuito consumo + alimentación	Circuito consumo + alimentación	35 horas
	Memoria EEPROM AT24C256	Memoria EEPROM AT24C256	10 horas
	Comunicación entre tarjetas núcleo	Comunicación entre tarjetas núcleo	30 horas
	Principal Máster	Principal Máster	10 horas
	Principal Slave	Principal Slave	5 horas
	Bajo consumo	Bajo consumo + Aitor + Álvaro	5 horas
	Desarrollo de la memoria	Desarrollo de la memoria	10 horas

9 ACRÓNIMOS UTILIZADOS.

Acrónimo	Significado
MCU	Microcontroller Unit (Unidad de Microcontrolador)
UART	Universal Asynchronous Receiver/Transmitter
GPIO	General Purpose Input/Output
PCB	Printed Circuit Board (Placa de Circuito Impreso)
RTD	Resistance Temperature Detector
RFID	Radio-Frequency Identification
EEPROM	Electrically Erasable Programmable Read-Only Memory
I ² C	Inter-Integrated Circuit (Protocolo de comunicación)
SPI	Serial Peripheral Interface
V _d	Tensión diferencial en la resistencia shunt
V _{out}	Tensión de salida del acondicionador
R _{shunt}	Resistencia de medida de corriente
G	Ganancia
AI	Amplificador Instrumental
AO	Amplificador Operacional
STM32	Familia de microcontroladores ARM Cortex-M de ST
LCD	Liquid Crystal Display
LED	Light Emitting Diode
SNTP	Simple Network Time Protocol
RTC	Real-Time Clock
RTOS	Real-Time Operating System
CMSIS	Cortex Microcontroller Software Interface Standard
UID	Unique Identifier (Identificador único)
DHT	Digital Humidity and Temperature sensor
BME680	Sensor ambiental (gas, humedad, temperatura)
BH1750	Sensor de luminosidad digital
HC-SR04	Sensor ultrasónico de distancia
LUX	Unidad de medida de iluminación

10 BIBLIOGRAFÍA UTILIZADA.

- [RD1] STMicroelectronics, *STM32F429xx Datasheet: ARM® Cortex®-M4 32b MCU+FPU, up to 2MB Flash/256+4KB RAM*, 2021. [Online]. Available: <https://www.st.com/resource/en/datasheet/stm32f429zi.pdf>
- [RD2] STMicroelectronics, *RM0090 Reference Manual: STM32F405/415, 407/417, 427/437 and 429/439 advanced ARM®-based 32-bit MCUs*, 2023. [Online]. Available: https://www.st.com/resource/en/reference_manual/dm00031020.pdf
- [RD3] STMicroelectronics, *UM1725: Description of STM32F4 HAL and Low-layer drivers*, Rev 4, Mar. 2017. [Online]. Available: https://www.st.com/resource/en/user_manual/dm00104801.pdf
- [RD4] STMicroelectronics, *UM1974: Getting started with STM32 Nucleo-144 board featuring STM32F429ZI MCU*, Rev 3, Nov. 2018. [Online]. Available: https://www.st.com/resource/en/user_manual/dm00244518.pdf
- [RD5] M. Bavari, *Mastering STM32*. Leanpub, 2017. [Online]. Available: <https://leanpub.com/mastering-stm32>
- [RD6] Platinum Resistance Temperature Sensors RS pro 2 Wire PT100 Sensor, -50°C Min +500°C Max, 5mm Probe Length X 2mm Probe Diameter. Available: <https://docs.rs-online.com/034e/A70000008838965.pdf>
- [RD7] NXP Semiconductors, *MFRC522: Standard Performance MIFARE and NTAG Frontend*, Rev. 3.9, Apr. 27, 2016. [Online]. Available: <https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf>
- [RD8] Handson Technology User Guide HC-SR04 Ultrasonic Sensor Module User Guide User Guide: Ultrasonic Sensor V2.0. Available: <https://www.handsontec.com/datasheets/HC-SR04-Ultrasonic.pdf>
- [RD9] “Tutorial Módulo Lector RFID RC522.” Naylamp Mechatronics - Perú, https://naylampmechatronics.com/blog/22_tutorial-modulo-lector-rfid-rc522.html
- [RD10] MaJerle. “Stm32f429/00-STM32F429_LIBRARIES/Tm_stm32f4_mfrc522.h at Main · MaJerle/Stm32f429.” GitHub, 2025, https://github.com/MaJerle/stm32f429/blob/main/00-STM32F429_LIBRARIES/tm_stm32f4_mfrc522.h. Accessed 24 June 2025.
- [RD11] Analog Devices. (2011). *OP07: Ultralow Offset Voltage Operational Amplifier (Rev. G)*. <https://www.analog.com/media/en/technical-documentation/data-sheets/op07.pdf>
- [RD12] Texas Instruments. (2013). *LMC7660: Switched Capacitor Voltage Converter (Rev. C)*. <https://www.ti.com/lit/ds/symlink/lmc7660.pdf>
- [RD13] Cadence Design Systems, Inc. (2000). *PSpice Library Guide OrCAD (Release 9.2)*. <https://www.cadence.com/>
- [RD14] Handson Technology. (n.d.). *HC-SR04 Ultrasonic Sensor Module User Guide (V2.0)*. <http://handsontec.com/index.php/product/hc-sr04-ultrasonic-ranging-module/>
- [RD15] ROHM Co., Ltd. (2011). *BH1750FVI: Digital 16bit Serial Output Type Ambient Light Sensor IC (Rev. D)*. <https://www.rohm.com/datasheet/BH1750FVI>
- [RD16] Bosch Sensortec. (2020). *BME680: Gas Sensor Combined with Humidity, Pressure and Temperature Sensors (BST-BME680-DS001-15)*. <https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme680-ds001.pdf>
- [RD17] Atmel. (2007). *AT24C128/256: Two-wire Automotive Temperature Serial EEPROMS (Rev. 5121B–SEEPR–2/07)*. <https://www.microchip.com/en-us/product/AT24C256>
- [RD18] Autor desconocido. (2025, abril 20). *Memoria EEPROM* [PDF]. Documento interno.
- [RD19] NXP Semiconductors. (2007). *MFRC522: Contactless Reader IC (Rev. 3.2)*. <https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf>