Carrera: Ingeniería Electrónica

Asignatura: Técnicas Digitales II

Año: 2024

Grupo N° 5

Alumnos:

- **Fernando Nahuel Sotelo**
- **Daniel Aragón**
- **Nahuel Enrique Beltrán**
- **José Gareca**

Profesor:

-Ing. Rubén Darío Mansilla

ATTP:

-Ing. Lucas Abdala

Informe de proyecto Final

Título del Proyecto Final: Sistema de Control Remoto de Actuadores con Monitoreo Ambiental

1. Consideraciones sobre el hardware del proyecto

1.1. Descripción del proyecto:

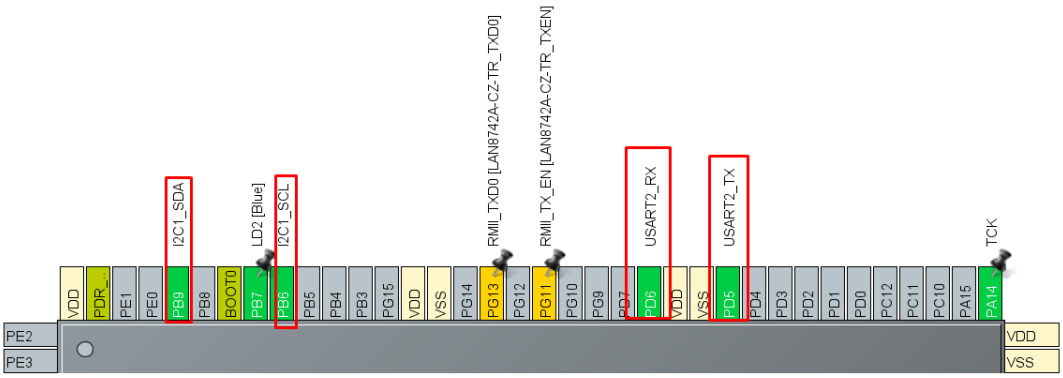
Esta aplicación implementa un sistema embebido basado en una placa STM32 que monitorea el nivel de luz ambiente mediante un fotosensor, controla la activación de LEDs y relés a través de comandos enviados por Bluetooth y muestra información relevante en una pantalla LCD I2C**.**
Plataforma Embebida: NUCLEO-STM32F429

El objetivo principal es crear un sistema interactivo que:
-Permita visualizar el porcentaje de luz ambiente detectado.
-Controle el encendido y apagado de tres leds de distintos colores mediante comandos remotos.
-Habilite o deshabilite tres relés para manejar cargas extras.
-Facilite la interacción mediante una interfaz bluetooth para enviar comandos, además de mostrar estados y mediciones en una pantalla LCD.

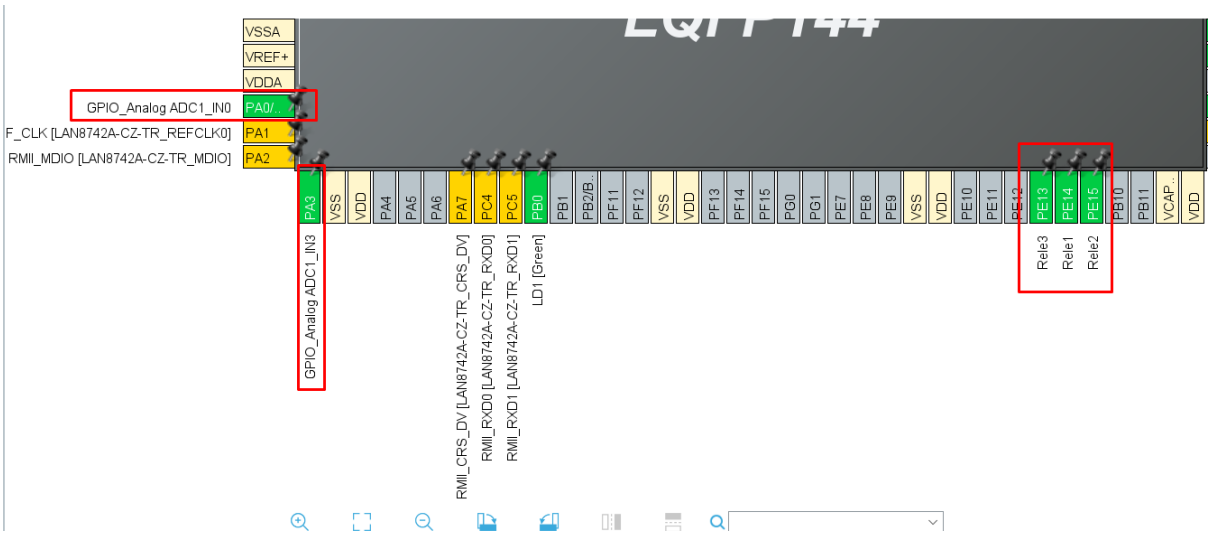
Funcionamiento
Medición de Luz Ambiente: El sistema lee continuamente el valor del fotosensor usando el ADC de la STM32. Calcula el porcentaje de luz y actualiza la pantalla LCD si hay una variación mayor al 5%.
Comunicación Bluetooth: Recibe comandos a través de UART2 conectada al módulo Bluetooth. Los comandos permiten encender o apagar LEDs y relés. Responde mediante UART confirmando cada acción realizada.
Control de LEDs y Relés: Los LEDs (verde, rojo y azul) se controlan con comandos específicos. Los relés se activan o desactivan para gestionar cargas externas, mostrando su estado en la LCD.
Interfaz de Usuario: La pantalla LCD muestra en tiempo real el porcentaje de luz detectado y el estado de los relés. Se brinda un mensaje de bienvenida al iniciar el sistema para indicar que está listo para recibir comandos.

1.2. Circuito del proyecto

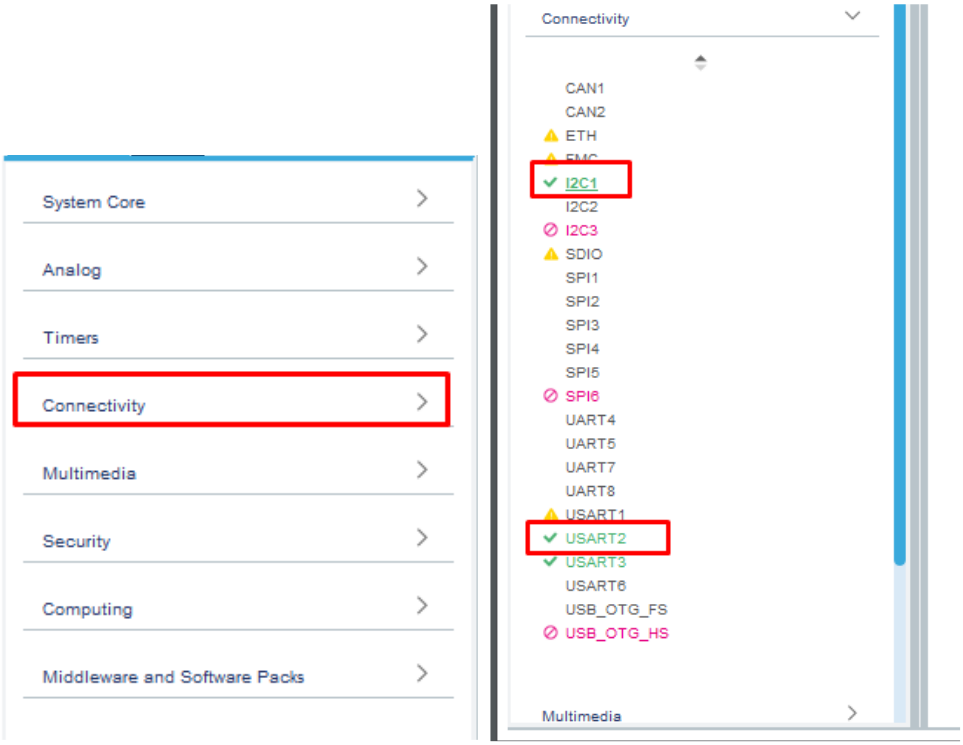
En este apartado veremos la configuración de los pines y conexionado físico:
Aquí se puede apreciar cómo están configurados los pines correspondientes a la comunicación **I2C** y **UART**



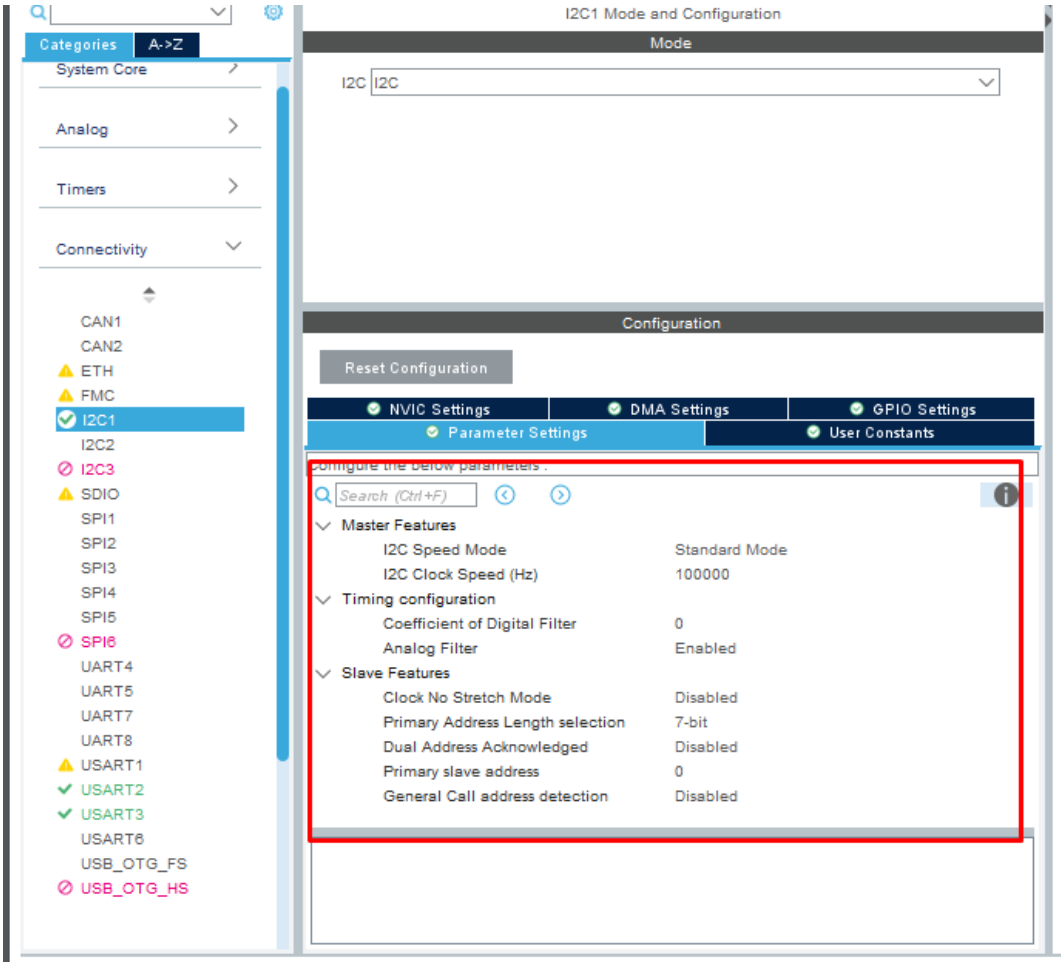
En esta otra podemos ver los pines asignados a los relés y al conversor ADC



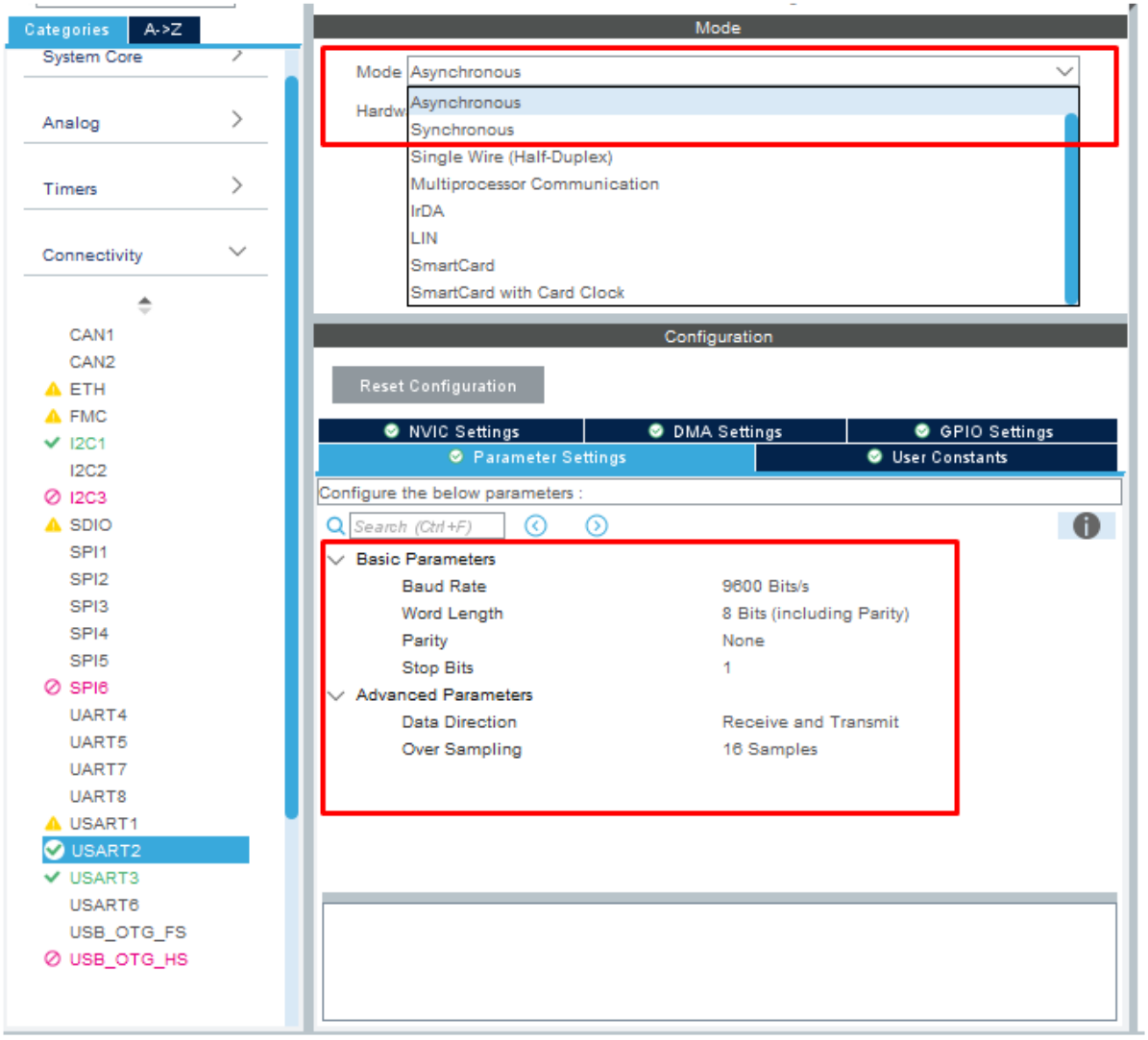
Ahora veremos la configuración de las comunicaciones:



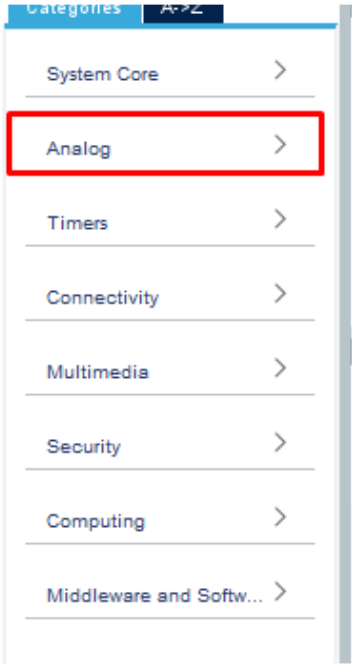
I2c:

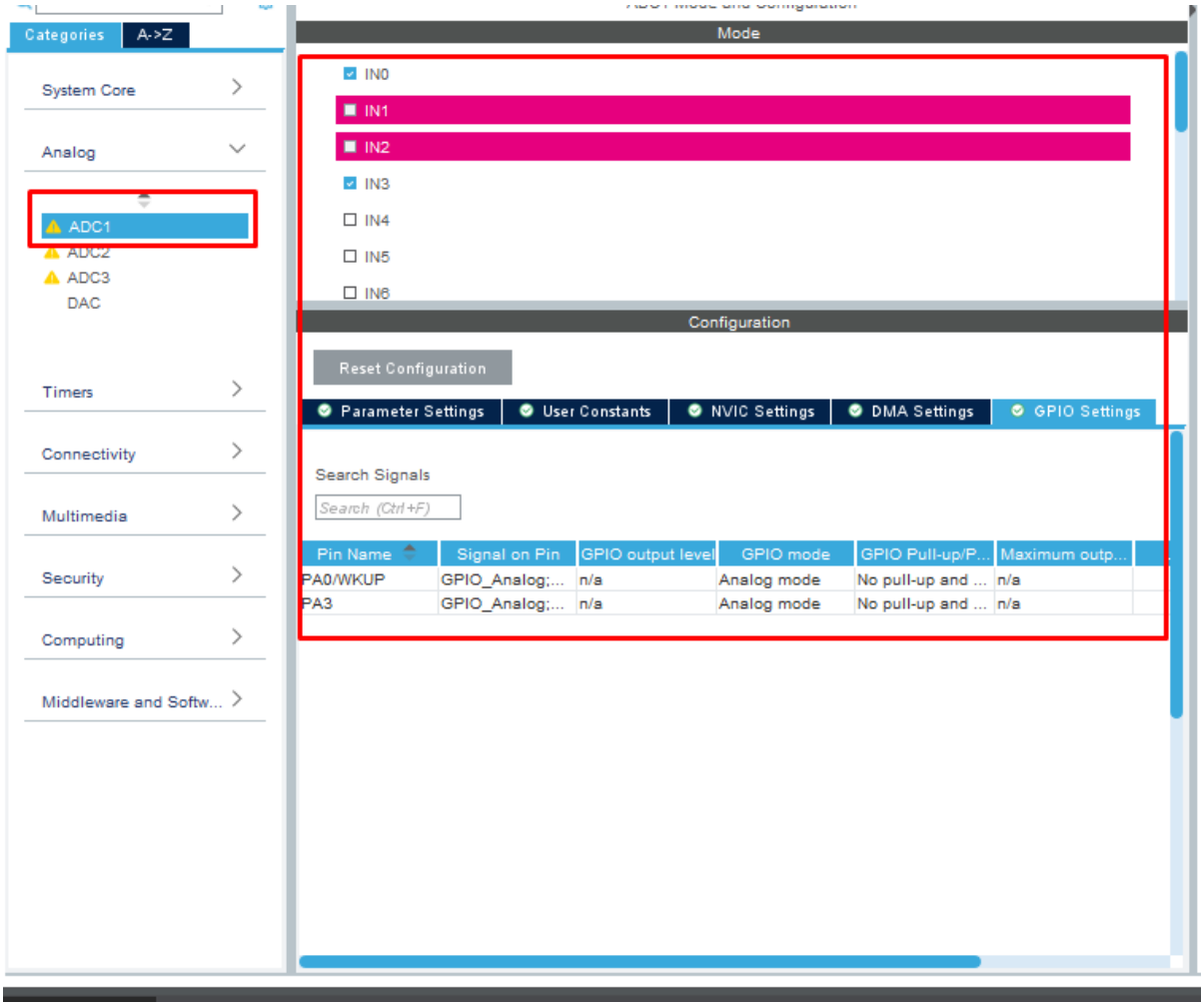


UART2:

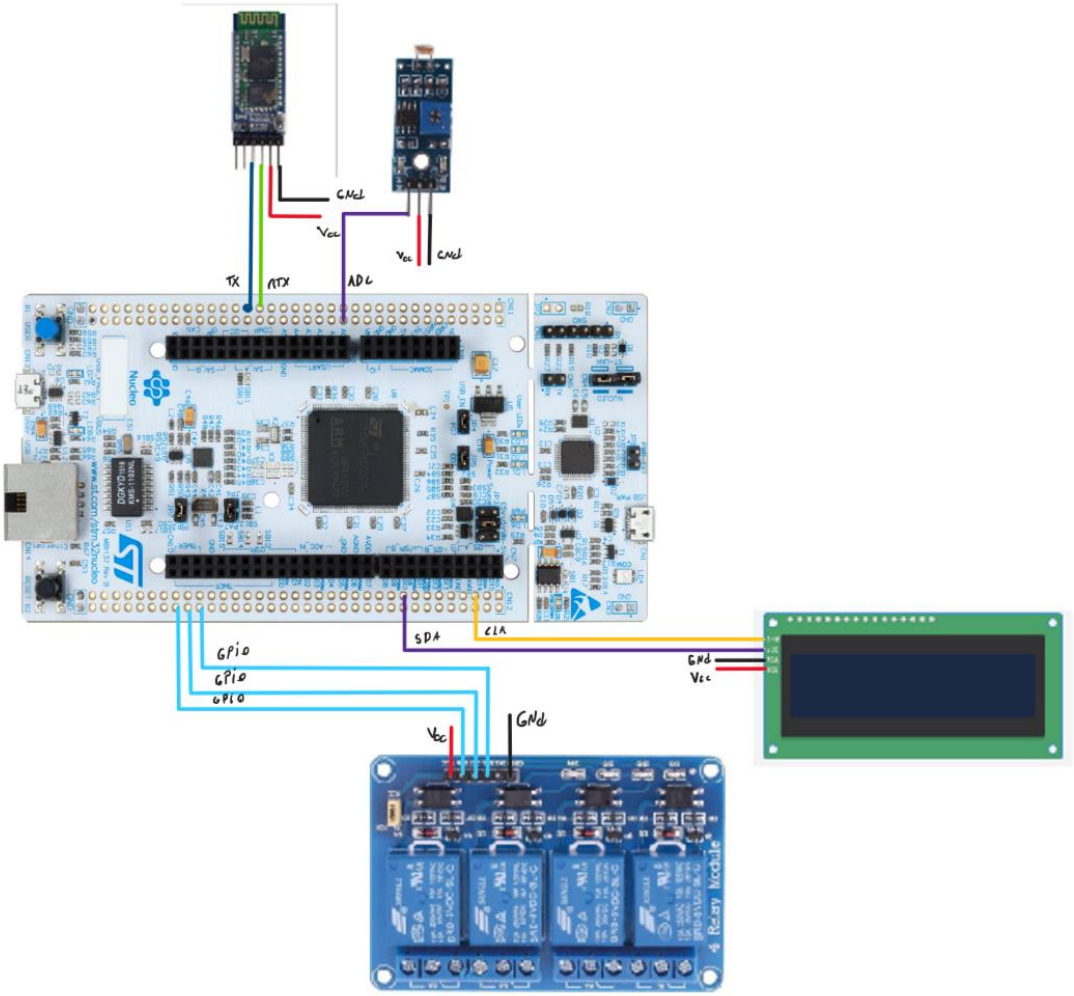


Ahora conversor adc:




Categories
A->Z
System Core
Analog
ADC1
ADC2
ADC3
DAC
Timers
Connectivity
Multimedia
Security
Computing
Middleware and Softw...
Mode
IN0
IN1
IN2
IN3
IN4
IN5
IN6
Configuration
Reset Configuration
Parameter Settings
User Constants
NVIC Settings
DMA Settings
GPIO Settings
Search Signals
Search (Ctrl+F)
Pin Name
Signal on Pin
GPIO output level
GPIO mode
GPIO Pull-up/P...
Maximum outp...
PA0/WKUP
GPIO_Analog;...
n/a
Analog mode
No pull-up and ...
n/a
PA3
GPIO_Analog;...
n/a
Analog mode
No pull-up and ...
n/a

Conexionado físico:



NOTA: se utilizará una fuente externa que se conecta en **VCC** y **GND**

1.3. Listado de componentes

-Módulo Bluetooth HC-05: Permite la comunicación inalámbrica mediante el protocolo Bluetooth, ideal para conexiones serie (UART) entre microcontroladores y otros dispositivos. Soporta modos maestro y esclavo.

-Módulo de 4 relés con optoacoplador: Placa que permite controlar hasta cuatro cargas de alta potencia (como motores o luces) mediante señales de bajo voltaje. Los optoacopladores aíslan eléctricamente el microcontrolador de la carga, aumentando la seguridad.

-Módulo LDR (fotorresistor): Sensor que varía su resistencia según la intensidad de luz recibida. Se usa comúnmente para medir niveles de iluminación o activar sistemas de manera automática en función de la luz ambiental.

-Nucleo-F429: Placa de desarrollo basada en el microcontrolador STM32F429, que ofrece alto rendimiento, procesamiento en tiempo real y múltiples interfaces de comunicación. Ideal para proyectos avanzados de control y procesamiento de datos.

-Pantalla 16x2 con interfaz I2C: Display LCD de 16 caracteres por 2 líneas, equipado con un módulo I2C que simplifica su conexión al microcontrolador al reducir el número

de pines necesarios. Se usa para mostrar información como mensajes, lecturas de sensores o estados del sistema.
-Fuente de 5v: Dispositivo que proporciona un voltaje estable de 5V para alimentar circuitos electrónicos. Es fundamental para asegurar un funcionamiento seguro y confiable de módulos y microcontroladores.

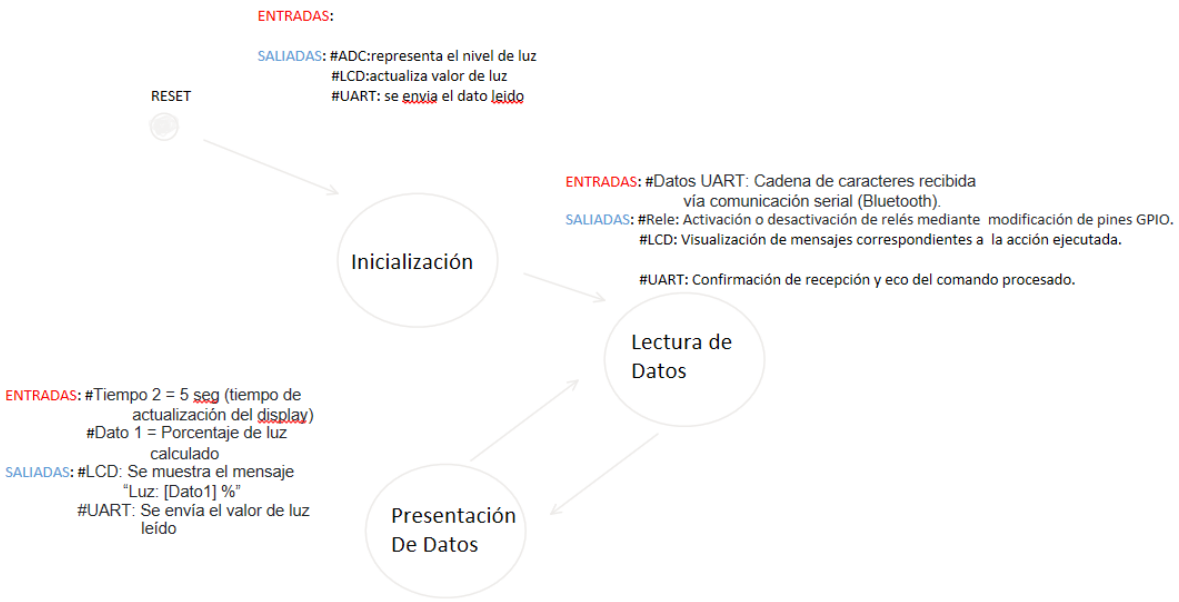
2. Consideraciones sobre el software

2.1. Link al repositorio

https://github.com/NahuelFSotelo/Grupo-1-TDII-AFP_2024.git

Nombre del Proyecto: trabajo-n°5

2.2. Descripción de funcionamiento de la aplicación del proyecto.



Lectura del Fotosensor:

Entradas: # No se requiere entrada externa directa (salvo la lectura del ADC a un sensor analógico).

Salidas: # Valor ADC: Representa el nivel de luz captado por el sensor.
#LCD: Se muestra el porcentaje de luz (%).
#UART: Se envía el dato leído vía comunicación serial.

Descripción: En este estado se realiza la conversión analógica-digital para obtener el nivel de luz del ambiente. La aplicación inicia el ADC mediante **HAL_ADC_Start()**,espera la conversióncon **HAL_ADC_PollForConversion()** y, al completarse, captura el valor a través de **HAL_ADC_GetValue()**, asignándolo a la variable photoSensorValue. Luego, detiene el ADC con **HAL_ADC_Stop()**. Posteriormente, se calcula el porcentaje de luz mediante la fórmula: **porcentaje = 100 - (photoSensorValue / 4095.0 * 100).**Si la diferencia con el valor previo supera

el umbral (5 unidades), se actualiza el mensaje en el Display LCD usando **lcd_enviar()** y se envía el dato por UART con **uartSendString().**

Tiempo de permanencia: Se evalúa y actualiza aproximadamente cada 500 ms, conforme al retardo establecido con **HAL_Delay(500).**

Lectura de Datos:

Entradas: # Datos UART: Cadena de caracteres recibida vía comunicación serial (Bluetooth).

Salidas: # LED: Encendido o apagado de LED según el comando recibido.

#Rele: Activación o desactivación de relés mediante modificación de pines GPIO.

#LCD: Visualización de mensajes correspondientes a la acción ejecutada.

#UART: Confirmación de recepción y eco del comando procesado.

Descripción: En este estado se revisa la presencia de datos recibidos por la función **uartGetReceivedData()**, la cual devuelve una cadena de caracteres. Si la longitud de la cadena es mayor a cero, se envía un mensaje de confirmación y se compara el contenido recibido con una serie de comandos predefinidos (por ejemplo, “Encender led verde”, “Apagar led azul”, “**RaN**”, “**RcFF**”, etc.). Según el comando, se realiza la acción correspondiente: se encienden o apagan los LED (mediante **writeLedOn_GPIO()** o **writeLedOff_GPIO()**) y se activan o desactivan los relés configurando el estado de los pines GPIO (**HAL_GPIO_WritePin()**). Además, se actualiza la pantalla LCD con el mensaje relativo al estado del relé y se envía un mensaje de confirmación por UART.

Tiempo de permanencia: Este estado se evalúa continuamente en el bucle principal, permitiendo la respuesta inmediata a cada comando recibido.

Presentación de Datos:

Entradas: #Tiempo 2 = 5 seg (tiempo de actualización del display)

Dato 1 = Porcentaje de luz calculado

Salidas: # LCD: Se muestra el mensaje “Luz: [Dato1] %”

#UART: Se envía el valor de luz leído

#(Opcional) LED: Ningún LED parpadea en este estado en particular

Descripción: En este estado se presenta el dato medido del sensor de luz en el Display LCD de 16x2. La aplicación utiliza el valor calculado (por ejemplo, “100 - (photoSensorValue / 4095.0 * 100)”) y, si la diferencia respecto a la lectura anterior es igual o mayor a 5 unidades, actualiza la visualización en el LCD mediante la función **lcd_enviar()** para mostrar el mensaje “Luz: [Dato1] %”. Simultáneamente, se envía este valor por UART utilizando **uartSendString()**, permitiendo la

monitorización remota. Este estado se mantiene durante un lapso de 5 segundos (controlado por la periodicidad del bucle principal y los retrasos establecidos) para asegurar una visualización estable del dato antes de volver a iniciar una nueva lectura.

2.3 Listado de los módulos de software desarrollados en el proyecto

2.4 Utilizo en la carpeta API los siguientes modulos:

En **API_adc.c** se desarrollan los prototipos de las siguientes funciones que se declaran en el correspondiente archivo *include*: **API_adc.h**

```
void MX_ADC1_Init(void);                    // Inicializa el periférico ADC1 con la
                                            // configuración predeterminada (resolución,
                                            // modo de escaneo, alineación, etc.)

void ADC_Start(void);                       // Inicia la conversión ADC en el canal
                                            // previamente configurado

void ADC_SetChannel(uint32_t channel);      // Configura el canal
                                            // especificado para la conversión ADC
                                            //(por ejemplo, ADC_CHANNEL_0)

uint32_t ADC_GetValue(void);                // Retorna el valor digital obtenido tras
                                            // la conversión ADC (valor de 12 bits u otra
                                            // resolución configurada)

bool ADC_IsConversionComplete(void);        // Indica si la conversión ADC
                                            // ha finalizado (true si completa, false en
                                            caso contrario)
```

Enlace al Código Fuente: https://github.com/NahuelFSotelo/Grupo-1-TDII-AFP_2024/blob/68d42adcac879963a6655aab3cce57ddd8324a32/trabajo-n%C2%B05/Grupo-1-TDII-AFP_2_2024-main/i2c-uart/Drivers/API/Src/API_adc.c#L1

En **i2c-lcd.c** se desarrollan los prototipos de las siguientes funciones que se declaran en el correspondiente archivo *include*: **i2c-lcd.h**

```
void lcd_init(void);                        // Inicializa el LCD y configura el periférico de
                                            // comunicación

void lcd_send_cmd(char cmd);                // Envía un comando (ej. configuración) al
                                            // LCD
```

```
void lcd_send_data(char data);    // Envía un único dato (carácter) al LCD

void lcd_enviar(char *string, int row, int col); // Envía una cadena al LCD en la
                                                 // posición especificada (fila y columna)

void lcd_send_string(char *str);  // Envía una cadena al LCD desde la posición
                                  // actual del cursor

void lcd_put_cur(int row, int col); // Posiciona el cursor en el LCD en la fila y
                                    // columna indicadas

void lcd_clear(void);             // Limpia el contenido del LCD
```

Enlace al Código Fuente: https://github.com/NahuelFSotelo/Grupo-1-TDII-AFP_2024/blob/68d42adcac879963a6655aab3cce57ddd8324a32/trabajo-n%C2%B05/Grupo-1-TDII-AFP_2_2024-main/i2c-uart/Drivers/API/Src/i2c-lcd.c#L1

En **API_uart.c** se encuentran desarrollados los prototipos de las siguientes funciones que se declaran en el correspondiente archivo *include*: **API_uart.h**

```
bool uartInit(void);                    // Inicializa la UART y retorna true si es
                                        // exitosa, false en caso contrario

void uartSendString(uint8_t *pstring);  // Envía una cadena de texto a través de
                                        // la UART

void uartSendStringSize(uint8_t *pstring, uint16_t size); // Envía una cadena
                                                          // de texto de tamaño
                                                          // específico por UART

void uartReceiveString(uint8_t *pstring, uint16_t size);  // Recibe una cadena
                                                          //de texto vía UART y la
                                                          //almacena en el buffer
                                                          // indicado

void MX_USART2_UART_Init(void);         // Inicializa el periférico USART2
                                        // para la comunicación UART

void uartReceiveCallback(void);         // Función de callback para gestionar
                                        // la recepción de datos por UART

uint8_t *uartGetReceivedData(void);     // Retorna un puntero a los datos
                                        // recibidos por UART
```

Enlace al Código Fuente: https://github.com/NahuelFSotelo/Grupo-1-TDII-AFP_2024/blob/68d42adcac879963a6655aab3cce57ddd8324a32/trabajo-n%C2%B05/Grupo-1-TDII-AFP_2_2024-main/i2c-uart/Drivers/API/Src/API_uart.c#L1

En **API_GPIO.c** se desarrollan los prototipos de las siguientes funciones que se declaran en el correspondiente archivo *include*: **API_GPIO.h** (No todas se usarán en el proyecto por lo que solo listaremos aquellas que se utilizarán en el mismo. Las funciones "***Debounce***" sólo se usarán en caso de agregar el pulsador de usuario al proyecto y se desarrollaron durante el periodo de cursado de Programación de Microcontroladores)

```
void MX_GPIO_Init(void);                // Inicializa la configuración de los pines GPIO
                                        // (entradas/salidas según aplicación)

void writeLedOn_GPIO(led_t LDx);        // Enciende el LED especificado

void writeLedOff_GPIO(led_t LDx);       // Apaga el LED especificado

void toggleLed_GPIO(led_t LDx);         // Cambia el estado del LED (si está
                                        // encendido lo apaga y viceversa)
```

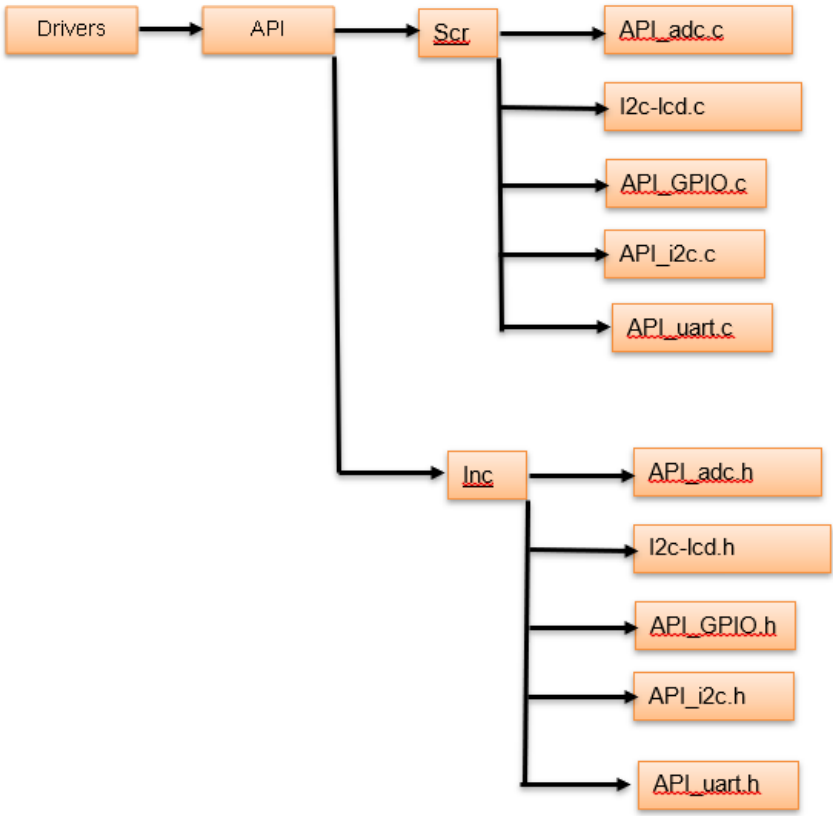
Enlace al Código Fuente: https://github.com/NahuelFSotelo/Grupo-1-TDII-AFP_2024/blob/68d42adcac879963a6655aab3cce57ddd8324a32/trabajo-n%C2%B05/Grupo-1-TDII-AFP_2_2024-main/i2c-uart/Drivers/API/Src/API_GPIO.c#L1

En **API_i2c.c** se encuentran desarrollados los comandos para introducir la configuración inicial del protocolo i2c que se declaran en el correspondiente archivo *include*: **API_i2c.h**

Enlace al Código Fuente: https://github.com/NahuelFSotelo/Grupo-1-TDII-AFP_2024/blob/68d42adcac879963a6655aab3cce57ddd8324a32/trabajo-n%C2%B05/Grupo-1-TDII-AFP_2_2024-main/i2c-uart/Drivers/API/Src/API_i2c.c#L1

La estructura de carpetas de los drivers a utilizar en el proyecto seria de la siguiente
manera:

Estructura de organización de carpetas de Drivers del proyecto:



3 Listado de los periféricos que utiliza en el proyecto.

Son los siguientes:

GPIO: Para manejo de los LEDs de usuario y la comunicación con el sensor ultrasónico.

I2C1: Para manejo de la comunicación con el LCD.

TIM1: Configurado para la implementación de un timer de 1 MHz en modo input capture, que permita manejar tiempos del orden de los μs para la medición de los datos y la comunicación con el sensor ultrasónico.

UART: Utilizado para la comunicación con el modulo HC-05

ADC: Convierte señales analógicas en señales digitales

4 Objetivos en la implementación del código

En el desarrollo del software del sistema embebido, se siguieron una serie de criterios para garantizar un código modular, eficiente y fácil de mantener. A continuación, se describen las principales buenas prácticas y objetivos de diseño que guiaron la implementación.

Modularidad y Organización del Código:

Uno de los principales objetivos fue estructurar el código de manera modular, separando funcionalidades en distintos archivos para mejorar la claridad y reutilización. Para ello, se definieron módulos específicos, tales como:

API_adc.c / API_adc.h: Funciones para la lectura del fotosensor mediante el ADC.

API_uart.c / API_uart.h: Manejo de la comunicación UART para la interacción con el módulo Bluetooth.

API_i2c.c / API_i2c.h: Comunicación con la pantalla LCD a través del protocolo I2C.

API_GPIO.c / API_GPIO.h: Control de LEDs y relés mediante pines GPIO.

Esta separación permitió que cada módulo fuese independiente y reutilizable en futuros proyectos, facilitando la depuración y actualización del código sin afectar otras partes del sistema.

Claridad y Legibilidad del Código

Para asegurar que el código sea comprensible y fácil de mantener, se adoptaron las siguientes prácticas:

Uso de **nombres descriptivos** para variables y funciones (por ejemplo, photoSensorValue, ADC_GetValue(), lcd_enviar()).

Estructura clara y ordenada con una separación lógica entre inicialización, ejecución de tareas y actualización de periféricos.

Inclusión de comentarios en secciones clave para explicar la lógica implementada.

Uso de **constantes y macros** en lugar de valores fijos, lo que facilita la modificación de parámetros sin alterar múltiples líneas de código.

Eficiencia en Sistemas Embebidos

Dado que el sistema embebido opera con recursos limitados, se implementaron estrategias para optimizar el rendimiento y evitar cargas innecesarias:

Minimización de cálculos dentro del bucle principal: Se evita recalcular valores constantemente y se actualizan solo cuando es necesario (por ejemplo, la pantalla LCD solo cambia si la lectura de luz varía en más de un 5%).

Uso de interrupciones en lugar de polling continuo: Se prioriza el uso de eventos para evitar bloqueos innecesarios.

Gestión de tiempos sin delays bloqueantes: Aunque en algunos casos se emplea HAL_Delay(), se busca migrar a temporizadores (TIM) para una ejecución más eficiente.

Robustez y Manejo de Errores

Un sistema embebido debe ser capaz de manejar errores y condiciones inesperadas. Para mejorar la robustez del software, se aplicaron las siguientes estrategias:

Confirmaciones de comandos vía UART: Se responde a cada comando recibido, asegurando que la acción se ejecutó correctamente.

Validación de datos de entrada: Se comprueba que las cadenas recibidas por Bluetooth correspondan a comandos válidos antes de ejecutarlos.

Verificación de la conexión de periféricos: Aunque no se implementó completamente, se considera agregar mecanismos para detectar fallos en la comunicación I2C y UART.

Uso de Estándares y Buenas Prácticas

Para garantizar la compatibilidad y facilitar futuras expansiones, se siguieron estándares de programación aplicados a sistemas embebidos:

Uso de la API HAL de STM32 para facilitar la portabilidad del código.

Definición de funciones con prefijos adecuados (API_, MX_) para distinguir entre inicialización de periféricos y funciones específicas del proyecto.

Organización en carpetas siguiendo una estructura clara, separando drivers, bibliotecas y archivos de aplicación.

4.1 Power of ten rules

Evitar control de flujo complejo como goto y recursiones:

No se observa el uso de goto.

No hay recursión en el código.

El flujo es lineal, basado en while y if.

Todos los bucles deben tener límites fijos.

El while(1) es un loop típico de sistemas embebidos. https://github.com/NahuelFSotelo/Grupo-1-TDII-AFP_2024/blob/68d42adcac879963a6655aab3cce57ddd8324a32/trabajo-n%C2%B05/Grupo-1-TDII-AFP_2_2024-main/i2c-uart/Core/Src/main.c#L128

No hay bucles for o while sin control adecuado.

Restringir la visibilidad de la información al mínimo posible.

Las variables globales son pocas, la mayoría son static o locales al main(). https://github.com/NahuelFSotelo/Grupo-1-TDII-AFP_2024/blob/68d42adcac879963a6655aab3cce57ddd8324a32/trabajo-n%C2%B05/Grupo-1-TDII-AFP_2_2024-main/i2c-uart/Drivers/API/Src/API_uart.c#L17

Buen manejo de encapsulación, ya que parte de la lógica está en las APIs (API_GPIO, API_uart, etc.).

Usar el preprocesador moderadamente.

Evitamos el procesamiento constante, por ejemplo:

pequeñas pausas entre procesos

actualizar la pantalla solo cuando hayan ocurrido cambios

https://github.com/NahuelFSotelo/Grupo-1-TDII-AFP_2024/blob/68d42adcac879963a6655aab3cce57ddd8324a32/trabajo-n%C2%B05/Grupo-1-TDII-AFP_2_2024-main/i2c-uart/Core/Src/main.c#L203

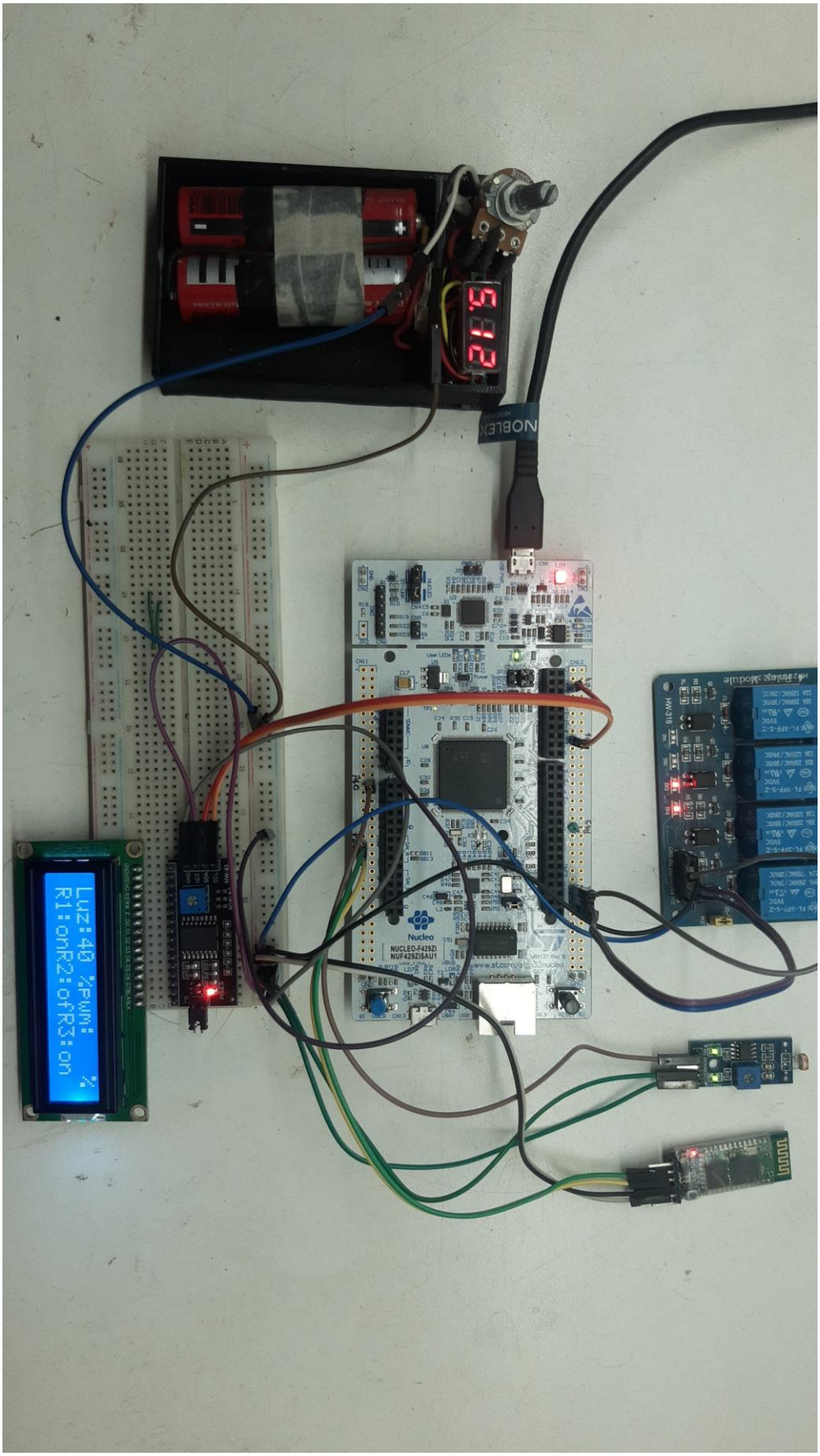
Conclusión

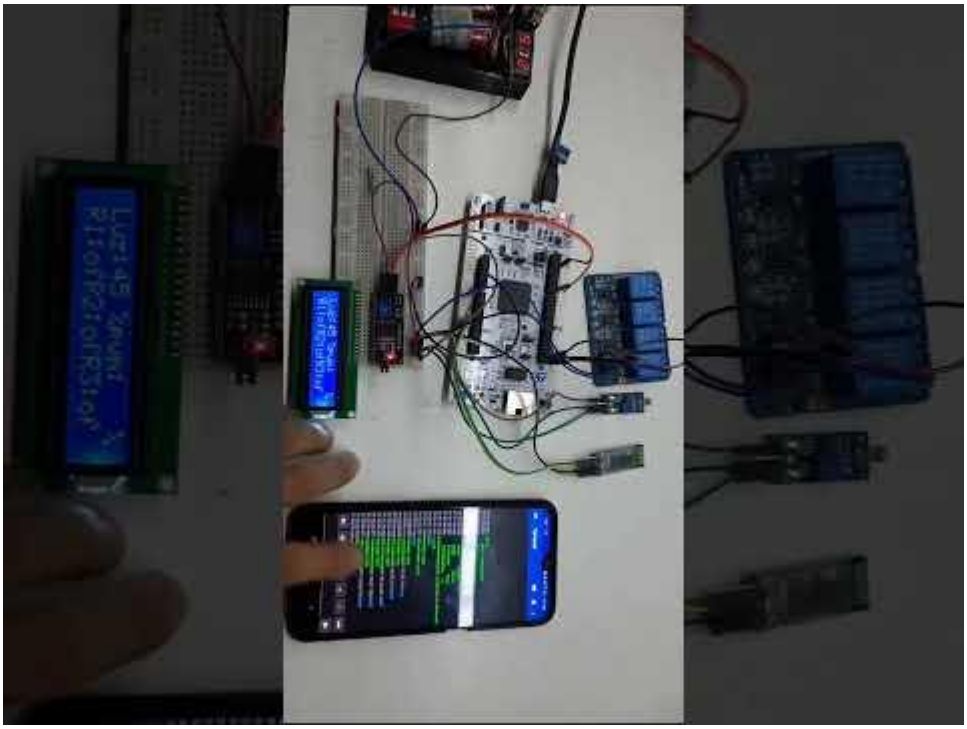
Siguiendo estos criterios, se logró un código estructurado, eficiente y mantenible, permitiendo futuras mejoras sin afectar el funcionamiento general del sistema. Aún hay aspectos a optimizar, como la migración de delays bloqueantes a temporizadores y una mejor gestión de errores en la comunicación con periféricos, pero la base establecida facilita la evolución del proyecto.

Enlace al código main:

https://github.com/NahuelFSotelo/Grupo-1-TDII-AFP_2024/blob/68d42adcac879963a6655aab3cce57ddd8324a32/trabajo-n%C2%B05/Grupo-1-TDII-AFP_2_2024-main/i2c-uart/Core/Src/main.c#L1

Aquí algunas imágenes del proyecto:





Enlace de video: https://youtube.com/shorts/CpNajV8YPLs?feature=share

5 datasheet

Núcleo F429Zi: https://www.st.com/en/evaluation-tools/nucleo-f429zi.html

LDR GL5528: https://pi.gate.ac.uk/pages/airpi-files/PD0001.pdf

Modulo relé: https://www.bolanosdj.com.ar/MOVIL/ARDUINO2/moduloRele.pdf

HC-05:https://components101.com/sites/default/files/component_datasheet/HC-05%20Datasheet.pdf

Display i2c: https://www.waveshare.com/w/upload/4/4d/LCD1602_I2C_Module.pdf