.UBAfiuba
FACULTAD DE INGENIERÍA

Trabajo Práctico Final

Programación de Microcontroladores
Carrera de Especialización en Sistemas Embebidos

Alumno:	Kirschner Lucas
Email:	kirschnerlucas1@gmail.com
Docente:	Ing. Patricio Boss

Índice

1. Introducción

El presente informe describe el desarrollo de un sistema de medición de temperatura y presión ambiental utilizando el sensor BMP280 de Bosch. Para la adquisición de los datos en tiempo real, se emplea la comunicación mediante el protocolo SPI. Los valores obtenidos son visualizados en un display LCD de 16x2 caracteres basado en el controlador HD44780. Con el objetivo de optimizar el uso de pines del microcontrolador, se utiliza un módulo adaptador PCF8574 que convierte la interfaz del display de paralelo a I2C.

El sistema se implementa sobre la plataforma NUCLEO-F446RE de STMicroelectronics, basada en un microcontrolador ARM Cortex-M4. El enfoque principal del trabajo se centra en el desarrollo de los drivers de comunicación para el sensor BMP280 (SPI) y para el display LCD (I2C), asegurando la correcta inicialización, adquisición de datos, procesamiento y visualización de la información ambiental de forma eficiente y modular.

La Figura 1 muestra la estructura general del sistema desarrollado. El microcontrolador se comunica con el sensor BMP280 mediante el protocolo SPI para adquirir datos de temperatura y presión. Paralelamente, utiliza una comunicación I2C para enviar los datos a un adaptador PCF8574, el cual convierte la señal para controlar el display LCD 16x2.

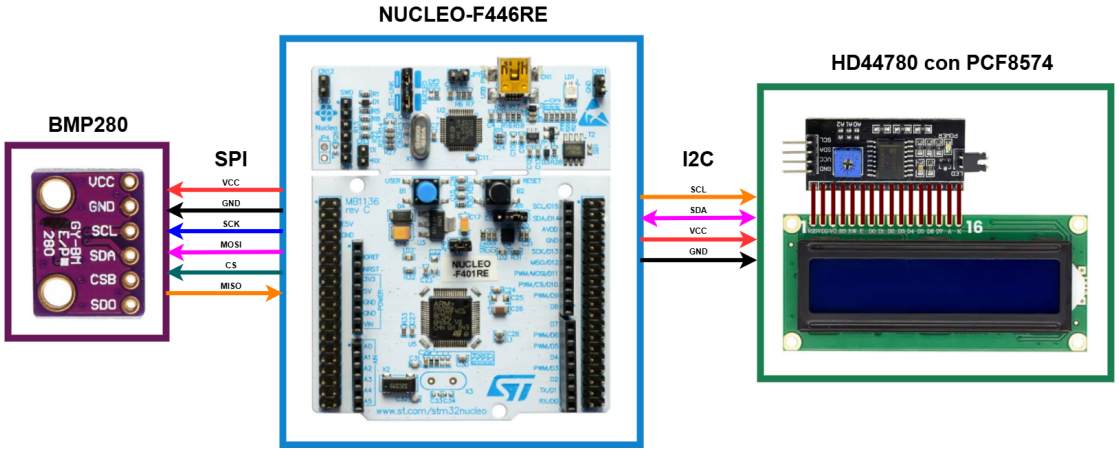

Figura 1: Diagrama de bloques del sistema.

2. Estructura del programa

El programa se organiza en módulos que encapsulan funcionalidades específicas, promoviendo una estructura clara y modular. La estructura general del programa es la que se muestra en la Figura 2.

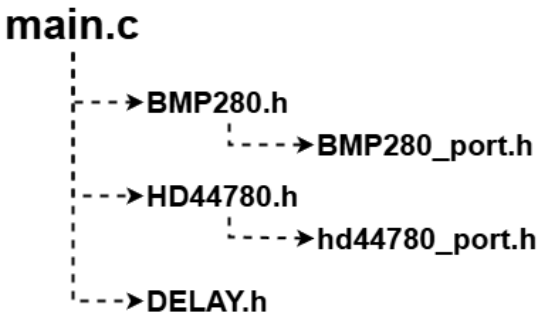

Figura 2: Estructura del programa.

2.1. Módulo main

El archivo main.c es el punto de entrada del programa. Se encarga de inicializar los periféricos del microcontrolador y de los dispositivos externos, como el sensor BMP280 y el display HD44780.

El control del sistema se organiza mediante una máquina de estados finitos, que gestiona las principales operaciones: iniciar mediciones, procesar datos, analizar los resultados y actualizar la interfaz de usuario. Además, implementa mecanismos de recuperación ante errores, alternando un LED de estado y reintentando la inicialización de los dispositivos.

2.2. Módulo BMP280

El módulo BMP280 se encarga de gestionar la interacción con el sensor de temperatura y presión BMP280. A través de este módulo, se inicializa el sensor, se configuran los parámetros de comunicación SPI, se disparan mediciones y se leen los datos compensados de temperatura y presión.

La comunicación se realiza mediante SPI, utilizando una capa de abstracción de hardware que facilita la portabilidad. El módulo incluye enumeraciones de estados, estructuras de datos y definiciones de registros.

2.2.1. Módulo BMP280_port

El módulo BMP280_port proporciona la capa de abstracción de hardware para el sensor BMP280. Gestiona los detalles específicos de la plataforma, como el uso del puerto SPI y el control de la señal Chip Select (CS). El objetivo de este módulo es desacoplar el driver del sensor de la implementación específica del hardware, facilitando la portabilidad a otros microcontroladores.

2.3. Módulo HD44780

El módulo HD44780 gestiona la comunicación con displays LCD basados en el controlador HD44780. Proporciona funciones para inicializar la pantalla, escribir caracteres, controlar la visualización de datos y posicionar el cursor, encapsulando la interacción con las instrucciones del display. Se apoya en una capa de abstracción de hardware encargada del manejo de las señales físicas.

2.3.1. Módulo HD44780_port

Este módulo encapsula las operaciones de bajo nivel necesarias para la comunicación con el display. En esta implementación, se utiliza un expansor de entradas/salidas PCF8574 conectado por I2C, reduciendo la cantidad de pines necesarios en el microcontrolador. Esta separación de responsabilidades permite que el módulo principal opere de manera independiente de la interfaz física utilizada. De este modo, con cambios mínimos en la capa de abstracción, es posible migrar la comunicación de una interfaz serie (I2C con PCF8574) a una conexión paralela directa con el display, sin modificar el resto del código de alto nivel.

2.4. Módulo DELAY

El módulo Delay gestiona retardos no bloqueantes utilizando el temporizador del sistema (SysTick). Proporciona una estructura de control y funciones asociadas para inicializar, modificar y verificar retardos sin detener la ejecución del programa.

Este enfoque es fundamental en drivers que operan mediante técnicas de polling, ya que permite administrar tiempos de espera o control de eventos temporizados sin bloquear el procesamiento de otras tareas.

3. Sensor de temperatura BMP280

3.1. Detalles del dispositivo BMP280

El BMP280 es un sensor barométrico absoluto de alta precisión desarrollado por Bosch [1], diseñado especialmente para aplicaciones móviles y dispositivos alimentados por batería.

Con un encapsulado compacto y comunicación mediante interfaces I2C o SPI, el BMP280 resulta ideal para sistemas embebidos que requieren mediciones precisas de presión. Ofrece una precisión relativa de ± 0.12 hPa (equivalente a ± 1 metro de altura) y una baja deriva térmica, lo que garantiza mediciones estables y confiables a lo largo del tiempo.

3.2. Módulo de bajo nivel BMP280_port

El módulo BMP280_port implementa la capa de abstracción de hardware necesaria para comunicar el microcontrolador STM32 con el sensor BMP280 mediante una interfaz SPI. Su principal objetivo es encapsular todos los detalles de la comunicación física, permitiendo que la lógica de más alto nivel permanezca independiente del hardware específico.

Esta estructura facilita la portabilidad del driver, ya que cualquier cambio en el periférico SPI, en el microcontrolador o incluso en el protocolo de comunicación (por ejemplo, pasando a I2C), puede ser manejado exclusivamente dentro de este módulo, sin afectar el resto del código.

3.2.1. Funcionalidades principales

Inicialización de la interfaz: La función BMP280_Port_Init() configura el periférico SPI3 como maestro, establece parámetros como la polaridad de reloj, fase, velocidad y modo de transferencia, utilizando la HAL de STM32. Esta inicialización permite garantizar una comunicación correcta con el BMP280.

Control del Chip Select (CS): Se incluyen funciones privadas para activar y desactivar el pin CS (BMP280_CS_Enable() y BMP280_CS_Disable()). Esto asegura que las transferencias SPI comiencen y finalicen correctamente, respetando las condiciones que el BMP280 requiere, como marcar el final de una lectura por ráfaga en el flanco ascendente del CS.

Operaciones de transferencia de datos:

- BMP280_Write(): transmite datos al sensor.
- BMP280_Read(): lee datos del sensor.
- BMP280_Transfer(): realiza una operación de transmisión y recepción simultánea.

Todas estas funciones manejan el estado del pin CS de forma automática y utilizan un timeout para asegurar que no queden transacciones bloqueadas.

Conversión de Estados: El estado devuelto por las funciones de la HAL es convertido a un tipo propio bmp280_port_status_t, mejorando la claridad de la interfaz hacia el resto del driver.

3.2.2. Consideraciones de portabilidad

El archivo BMP280_port.h permite ajustar fácilmente:

- El periférico SPI utilizado (hspi3).
- El pin y puerto del Chip Select (SPI3_CS_Pin y SPI3_CS_GPIO_Port).
- La inclusión del HAL correspondiente (stm32f4xx_hal.h), que puede modificarse para adaptarse a otra familia de STM32 u otra plataforma.

3.3. Módulo de alto nivel BMP280

El driver de alto nivel implementado en los archivos BMP280.h y BMP280.c proporciona una interfaz completa para la comunicación con el sensor BMP280 utilizando las funciones de bajo nivel descriptas en la Sección 3.2. Este módulo permite la inicialización del sensor, la configuración de parámetros de medición, la lectura de datos crudos de temperatura y presión, y la compensación de estos valores aplicando las constantes de calibración internas.

3.3.1. Funcionalidades públicas

El archivo BMP280.h expone al usuario del driver lo siguiente:

Macros de definición: Mapas de memoria de registros, modos de operación, coeficientes de filtro, y máscaras para operaciones SPI de lectura/escritura.

Enumeraciones: Define bmp280_status_t, una enumeración de estados que reportan el resultado de las operaciones del driver (por ejemplo, BMP280_OK, BMP280_ERROR_COMM, etc.).

Estructuras de datos: Define bmp280_t, una estructura que almacena los valores de temperatura y presión compensados.

Prototipos de funciones públicas:

- BMP280_Init(): Inicializa y verifica la presencia del sensor.
- BMP280_Trigger_Measurement(): Dispara una medición en modo forzado.
- BMP280_Is_Measuring(): Verifica si el sensor sigue en proceso de medición.
- BMP280_Update_Parameters(): Lee los registros crudos, aplica la compensación y actualiza los datos de temperatura y presión.

3.3.2. Funcionalidades privadas

El archivo BMP280.c implementa la lógica del driver, incluyendo:

Manejo de calibración interna:

- Se define una estructura privada bmp280_calib_data_t para almacenar los coeficientes de calibración del sensor.
- La función privada BMP280_Get_Calibration() lee los registros de calibración de forma eficiente mediante una única transferencia SPI en ráfaga (25 bytes).

Implementación de funciones principales:

- BMP280_Init(): Verifica la identidad del sensor, aplica un reset de software si es necesario, y configura parámetros básicos como oversampling y modo de medición. También lee los coeficientes de calibración para preparar la compensación de datos.
- BMP280_Trigger_Measurement(): Permite disparar manualmente una nueva medición en modo forzado, útil para reducir el consumo energético del sistema.
- BMP280_Is_Measuring(): Revisa el bit de estado del sensor (STATUS_MEASURING) para determinar si los datos están listos para ser leídos.

- BMP280_Update_Parameters(bmp280_t* dev): Realiza una lectura de los registros crudos de temperatura y presión. Aplica la compensación utilizando las fórmulas proporcionadas en la hoja de datos del sensor BMP280. Actualiza la estructura bmp280_t con los valores finales en unidades físicas (°C y Pa).

Consideraciones adicionales:

- Se presta especial atención a respetar los tiempos de medición y el manejo correcto del protocolo SPI.
- Se incluye un API_swo.h para impresión de mensajes de debug en tiempo real, facilitando la validación del correcto funcionamiento del driver.
- Se evita el uso de 'magic numbers' (valores hardcodeados) mediante el uso de macros descriptivos.

4. Display HD44780 con adaptador PCF8574

4.1. Detalles del display HD44780

El HD44780U es un circuito integrado diseñado para controlar pantallas LCD de matriz de puntos que despliegan caracteres alfanuméricos, símbolos y caracteres [2]. Incorpora en un solo chip todas las funciones necesarias para operar un display: RAM de visualización (DDRAM), generador de caracteres (CGROM/CGRAM) y controladores de señal para el LCD. Admite comunicación en modo 4 bits o 8 bits con un microprocesador externo, permitiendo sistemas de conexión simplificados.

4.2. Detalles del adaptador serie-paralelo PCF8574

El PCF8574 es un expansor de entrada/salida de 8 bits que se comunica a través del bus I^2C [3]. Está pensado para ampliar la cantidad de pines disponibles en un microcontrolador usando sólo dos líneas: SCL (reloj) y SDA (datos).

Cada pin del PCF8574 (P0 a P7) es de tipo quasi-bidireccional: pueden actuar como entrada o salida sin necesidad de configurar explícitamente una dirección de datos. Al encenderse, los pines están en estado alto, lo que permite configuraciones seguras para aplicaciones como manejo de LEDs o, en este caso, control de un LCD.

En este proyecto, el PCF8574 actúa como un puente I^2C-paralelo para manejar las señales de control y datos del display HD44780 usando solamente dos pines del microcontrolador.

4.3. Módulo de bajo nivel HD44780_port

El módulo HD44780_port implementa una capa de bajo nivel para controlar un display basado en el controlador HD44780 mediante una interfaz I2C utilizando un expansor de E/S tipo PCF8574.

Su propósito es abstraer la lógica de comunicación de 4 bits sobre I2C, de modo que el driver de más alto nivel (HD44780.c) pueda utilizar funciones simples para enviar comandos y datos al LCD, sin preocuparse por los detalles del protocolo ni del hardware específico.

4.3.1. Funcionalidades principales

Inicialización de la interfaz I2C: La función HD44780_Port_Init() configura el periférico I2C del microcontrolador utilizando la biblioteca STM32 HAL para permitir la comunicación con el PCF8574.

Retardos de tiempo: HD44780_Port_Delay() proporciona un retardo bloqueante en milisegundos necesario para cumplir con los requisitos de temporización del HD44780.

Operaciones de transferencia de datos:

- HD44780_Port_Send_Nibble(): transfiere 4 bits de datos (nibble) hacia el LCD a través del expansor I2C, controlando también las señales de RW (Read/Write), RS (Register Select), EN (Enable) y Backlight (BL)
- HD44780_Port_Send_Byte(): divide un byte en dos nibbles y los envía secuencialmente utilizando la función HD44780_Port_Send_Nibble().

4.3.2. Consideraciones de portabilidad

El archivo HD44780_port.h permite ajustar fácilmente:

- El periférico I2C utilizado (hI2C1).
- La inclusión del HAL correspondiente (stm32f4xx_hal.h), que puede modificarse para adaptarse a otra familia de STM32 u otra plataforma.

4.4. Módulo de alto nivel HD44780

Esta interfaz proporciona un conjunto de funciones de alto nivel para el manejo de un display LCD basado en el controlador HD44780, utilizando comunicación de 4 bits. Utiliza las funciones de bajo nivel descriptas en la Sección 4.3, permitiendo operaciones como inicialización manual, envío de comandos, escritura de caracteres y configuración de parámetros de visualización.

4.4.1. Funcionalidades públicas

El archivo HD44780.h expone al usuario del driver lo siguiente:

Macros de definición: Define instrucciones del set de comandos del HD44780 y macros generadoras de comandos. También proporciona opciones predefinidas para configuración de entrada, control de display, desplazamientos y ajuste de modo de interfaz.

Enumeraciones: Define hd44780_status_t, una enumeración de estados que reportan el resultado de las operaciones realizadas por el driver.

Prototipos de funciones públicas:

- HD44780_Init(): Inicializa el controlador LCD ejecutando la secuencia de inicialización.
- HD44780_Write(): Escribe una cadena de caracteres en lam posisicón actual del cursor.
- HD44780_Write_int(): Escribe un valor entero en la posición actual del cursor.
- HD44780_Set_Cursor(): Posiciona el cursor en una fila y columna específicas del display.
- HD44780_Clear(): Limpia el contenido del display y resetea el cursor a su posición inicial.

4.4.2. Funcionalidades privadas

El archivo HD44780.c implementa la lógica de funcionamiento del controlador, incluyendo:

Implementación de funciones principales:

- HD44780_Init(): Ejecuta la secuencia de inicialización descrita en el datasheet, aplicando pulsos de configuración, esperas de tiempos críticos, y selección de modo de interfaz (por ejemplo, 4 bits, 2 líneas, fuente de 5x8 puntos).
- HD44780_Write(): Gestiona la trasnmisión de strings, finalizados con el caracter nulo.
- HD44780_Write_int(): Convierte un entero de con signo de 16 bits en una cadena de carateres y la escribe en la dirección actual.
- HD44780_Set_Cursor(): Calcula y posiciona el cursor en la la dirección de DDRAM correspondiente a la fila y columna especificadas.
- HD44780_Clear(): Encapsula el envío del comando que borra y ubica el cursor en la primera posición y espera el tiempo necesario luego de su ejecución.

Consideraciones adicionales:

- Todas las operaciones consideran los tiempos mínimos de espera entre comandos según las especificaciones del HD44780.
- Se utilizan las funciones de bajo nivel de HD44780_port para desacoplar la implementación física del controlador.
- Se minimiza el uso de valores mágicos 'magic' numbers mediante el uso de macros descriptivos.

5. Aplicación

La capa de aplicación del proyecto se estructura en torno a una máquina de estados finita (FSM) que coordina la interacción entre el sensor de temperatura BMP280 y el display LCD HD44780. Esta máquina de estados ejecuta un ciclo periódico para adquirir, procesar, analizar y mostrar datos de temperatura y presión ambiental, manejando además posibles errores de inicialización o de comunicación.

5.1. Estructura general

El programa define un tipo enumerado state_t, que representa los distintos estados de la FSM:

- INIT_COMPONENTS: Inicializa el sensor BMP280, el display LCD y las interfaces SPI e I2C.
- START_MEASUREMENT: Solicita al BMP280 el inicio de una medición en modo 'Forced'.
- WAIT_MEASUREMENT: Espera hasta que el sensor complete la medición, revisando un bit de estado del sensor.
- PROCESS_DATA: Lee los datos crudos del sensor y realiza la compensación con los valores de calibración para obtener la temperatura real.
- ANALYZE_DATA: Evalúa si la temperatura medida se encuentra fuera del rango esperado, definido por TEMP_MIN_C y TEMP_MAX_C .
- DISPLAY_DATA: Muestra los valor de temperatura y presión en el LCD, incluyendo una advertencia '(!)' si la temperatura se encuentra fuera de rango.
- WAIT_TIME: Introduce un retardo no bloqueante para fijar la periodicidad del ciclo de medición.

- ERROR_STATE: Indica la existencia de un error y gestiona la recuperación periódica invocando a la función de inicialización.

El estado inicial es INIT_COMPONENTS, y el flujo de ejecución se basa en transiciones condicionales en función de los resultados de las operaciones realizadas en cada estado.

5.2. Descripción de funciones

- FSM_Init(): Inicializa las variables internas de la FSM, los retardos de control y coloca al sistema en el estado de inicio.
- FSM_Update(): Implementa la lógica de transición entre estados. Se invoca de forma continua en el bucle principal del programa.

5.3. Manejo de errores

Si en cualquier punto ocurre una falla en la inicialización de los periféricos o en la comunicación con los dispositivos, la FSM transita hacia ERROR_STATE. En este estado:

- Se realiza el parpadeo del LED verde LD2 de la placa Nucleo-F446RE.
- Se espera un tiempo definido antes de intentar una nueva inicialización, invocando a la función FSM_Init().

5.4. Variables de retardo

Se emplea una estructura delay_t para controlar los retardos sin bloquear el flujo de ejecución:

- delayFSM define el tiempo entre ciclos normales de la FSM.
- delayLED controla la frecuencia del parpadeo en caso de error.
- delayReinit define la espera antes de intentar una recuperación automática.

6. Implementación de buenas prácticas de programación

El desarrollo de este trabajo incorpora múltiples buenas prácticas de programación, alineadas tanto a principios generales de robustez y eficiencia como a las recomendaciones específicas de "The Power of Ten Rules for Developing Safety Critical Code"[4].

Utilización de variables privadas: Cuando una variable global es necesaria dentro de un módulo, se recomienda declararla como static para que sea privada al archivo en el que se encuentra. A modo de ejemplo, se define la estructura bmp280_calib_data_t que almacena los coeficientes de calibración del sensor BMP280. Esta información es necesaria internamente para calcular correctamente la temperatura y presión, pero no es relevante ni accesible para el usuario de la API. Por lo tanto, se declara una variable global privada utilizando la palabra clave static (https://github.com/lucaskirschner/PdM-practicas/blob/a7d72c169a7a46a1ed6965cec1eed66bc5bacc2c/Trabajo_Final/BMP280_Display_Station/Core/API/BMP280/Src/BMP280.c#L32).

Abstracción del Hardware: Toda la comunicación de bajo nivel está desacoplada del driver y encapsulada en los módulos correspondientes, favoreciendo la portabilidad hacia diferentes plataformas o MCUs.

Implementación de una Máquina de Estados Finitos (MEF): El uso de una Máquina de Estados Finitos (MEF) es una excelente práctica en el desarrollo de sistemas embebidos debido a su capacidad para organizar el flujo de ejecución del programa en función de eventos o condiciones específicas. Al dividir el comportamiento del sistema en estados claramente definidos, la MEF facilita la lectura, mantenibilidad y extensibilidad del código (https://github.com/lucaskirschner/PdM-practicas/blob/a7d72c169a7a46a1ed6965cec1eed66bc5bacc2c/Trabajo_Final/BMP280_Display_Station/Core/Src/main.c#L266).

Robustez: Las funciones devuelven un estado explícito que facilita el manejo de errores. Esto evita la propagación silenciosa de fallos, y responde a la regla de 'Check the return value of all non-void functions' (https://github.com/lucaskirschner/PdM-practicas/blob/a7d72c169a7a46a1ed6965cec1eed66bc5bacc2c/Trabajo_Final/BMP280_Display_Station/Core/API/HD44780/Inc/HD44780.h#L138).

Modularidad y escalabilidad: Los drivers están estructurados para permitir fácilmente la incorporación de nuevas funciones sin alterar el funcionamiento básico. Se promueve la expansión manteniendo la compatibilidad.

Documentación clara: Se utilizó documentación estilo Doxygen para describir cada función, estructura y módulo. Esta práctica facilita la generación de documentación técnica automática y mejora la mantenibilidad del proyecto.

Evitar bucles infinitos: Todas las funciones que involucran espera activa, como la máquina de estados, utilizan timeouts apropiados mediante el módulo Delay. De esta forma, se evita el bloqueo indeterminado, respetando la regla de .^All loops must have fixed bounds". De igual manera, las funciones de trasnferencia de datos mediante SPI e I2C utilizan timeouts adecuados para evitar bloqueos indeterminados, lo que ocurre al usar HAL_MAX_DELAY (https://github.com/lucaskirschner/PdM-practicas/blob/a7d72c169a7a46a1ed6965cec1eed66bc5bacc2c/Trabajo_Final/BMP280_Display_Station/Core/API/BMP280/Src/BMP280_port.c#L81).

Minimizar el uso de variables globales: Se restringió el ámbito de las variables a la mínima extensión posible. Variables como calib, que almacenan los coeficientes de calibración, fueron declaradas como estáticas dentro del módulo BMP280.c, evitando su exposición global.

Funciones limitadas en tamaño: Cada función del proyecto fue diseñada para caber dentro de una única página de impresión, respetando la regla de Restrict functions to a single printed page". Esto mejora la legibilidad y reduce la complejidad de comprensión (https://github.com/lucaskirschner/PdM-practicas/blob/a7d72c169a7a46a1ed6965cec1eed66bc5bacc2c/Trabajo_Final/BMP280_Display_Station/Core/API/BMP280/Src/BMP280.c#L43.).

Uso controlado de punteros: El uso de punteros fue limitado a una única desreferenciación y se evitó el uso de punteros a funciones, siguiendo la regla de 'Limit pointer use to a single dereference' (https://github.com/lucaskirschner/PdM-practicas/blob/a7d72c169a7a46a1ed6965cec1eed66bc5bacc2c/Trabajo_Final/BMP280_Display_Station/Core/API/HD44780/Src/HD44780.c#L106).

Comentarios que explican el "por qué": Se incluyeron comentarios enfocados en explicar la razón detrás de decisiones importantes de implementación (https://github.com/lucaskirschner/PdM-practicas/blob/a7d72c169a7a46a1ed6965cec1eed66bc5bacc2c/Trabajo_Final/BMP280_Display_Station/Core/API/HD44780/Src/HD44780.c#L24).

7. Conclusiones

Se concluye que se llevó a cabo con éxito el desarrollo del trabajo práctico final integrador, cumpliendo con los objetivos propuestos. Se implementaron dos drivers genéricos para el manejo de periféricos, logrando una solución modular y escalable.

Durante el desarrollo, se aplicaron de manera efectiva los conocimientos adquiridos sobre protocolos de comunicación y diseño de device drivers. Se configuró exitosamente el sensor BMP280 y se implementó la lectura de datos de temperatura y presión utilizando el protocolo SPI. Posteriormente, los datos obtenidos fueron presentados en un display LCD de 16x2 caracteres, basado en el controlador HD44780 y comunicado a la plataforma Nucleo-F446RE mediante el protocolo I2C, utilizando un adaptador serie-paralelo PCF8574.

La programación se realizó aplicando buenas prácticas de desarrollo de software embebido, tales como la separación de capas, la verificación sistemática de errores y el uso de documentación clara. Esto permitió no solo facilitar el proceso de implementación, sino también garantizar la mantenibilidad y escalabilidad de los drivers para futuras expansiones o adaptaciones.

En síntesis, el proyecto permitió consolidar los conocimientos teóricos y prácticos adquiridos en los cursos Protocolos de Comunicación en Sistemas Embebidos y Programación de Microcontroladores, enfrentando desafíos reales de integración de hardware y software, y sentando bases sólidas para desarrollos de mayor complejidad en sistemas embebidos.

Referencias

[1] Bosch Sensortec, "BMP280 – Digital pressure sensor," [Online]. Available: https://www.bosch-sensortec.com/products/environmental-sensors/pressure-sensors/bmp280/[Accessed: 21-Apr-2025].

[2] Hitachi, "HD44780U (LCD-II) Dot Matrix Liquid Crystal Display Controller/Driver," [Online]. Available: https://www.sparkfun.com/datasheets/LCD/HD44780.pdf. [Accessed: 21-Apr-2025].

[3] NXP Semiconductors, "PCF8574; PCF8574A Remote 8-bit I/O expander for I2C-bus with interrupt," [Online]. Available: https://www.nxp.com/docs/en/data-sheet/PCF8574_PCF8574A.pdf. [Accessed: 21-Apr-2025].

[4] J. Ganssle, "The Power of Ten—Rules for Developing Safety Critical Code," [Online]. Available: https://ieeexplore.ieee.org/document/8471073. [Accessed: 21-Apr-2025].

[5] L. Kirschner, "PdM-Prácticas," [Online]. Available: https://github.com/lucaskirschner/PdM-practicas.git. [Accessed: 21-Apr-2025].