Técnicas digitales II

Actividad de formacion practica 2

Profesores:
Ing. Rubén Darío Mansilla

ATTP:
Ing. Lucas Abdala

Grupo 5

Integrantes:

- Angelillo Alessandro
- Hualampa Leonel Javier
- Plaate Ivan
- Torres Juan

Introducción: utilización de drivers en Apps desarrolladas para sistemas embebidos

En el desarrollo de aplicaciones para sistemas embebidos, la utilización de **drivers** resulta una práctica fundamental para garantizar un diseño modular, escalable y de fácil mantenimiento. Un driver es una capa de software que se encarga de abstraer el acceso al hardware, ofreciendo funciones bien definidas para que la aplicación pueda interactuar con los periféricos sin necesidad de manipular directamente los registros del microcontrolador.

Las principales **ventajas de desarrollar aplicaciones utilizando drivers** son:

- **Abstracción del hardware:** la lógica de la aplicación no depende de detalles específicos de configuración de registros, sino de funciones de más alto nivel.
- **Reutilización de código:** un driver correctamente diseñado puede emplearse en múltiples proyectos sin modificaciones significativas.
- **Mantenibilidad y escalabilidad:** al separar la lógica de la aplicación del acceso al hardware, las actualizaciones o cambios en el microcontrolador afectan únicamente al driver y no a toda la aplicación.
- **Legibilidad y organización:** el código queda más ordenado.

La **creación y desarrollo de un driver** suele seguir una serie de pasos bien definidos.

En primer lugar, se realiza la **definición del alcance**, donde se decide qué periférico o recurso del microcontrolador se va a controlar y qué funciones debe ofrecer el driver.

Luego, se pasa a la **creación de la estructura del driver**, que generalmente está compuesta por dos archivos:

- Un archivo **.h (header o cabecera)**, donde se declaran los prototipos de las funciones, los tipos de datos, constantes o macros que serán visibles desde la aplicación.
- Un archivo **.c**, donde se escribe la implementación de esas funciones, accediendo al hardware mediante las librerías del fabricante (como HAL en STM32) o directamente mediante los registros del microcontrolador.

A continuación, se procede a la **programación del driver**, que consiste en desarrollar las funciones previamente definidas en el archivo de cabecera. En esta etapa es importante ser consistente en los parámetros que se utilizan, validar posibles errores y asegurarse de que el periférico esté correctamente inicializado antes de usarse.

El siguiente paso es la **documentación del driver**, que implica agregar comentarios claros en el código para explicar qué hace cada función, qué parámetros recibe y qué valores devuelve. Una buena práctica es usar un formato estandarizado, lo que facilita generar documentación automática y asegura que cualquier programador pueda entender y reutilizar el driver en el futuro.

Finalmente, se lleva a cabo la **prueba y verificación**, donde el driver se integra en una aplicación de ejemplo para comprobar que todas sus funciones trabajen como se espera. Aquí también conviene verificar situaciones no previstas, como parámetros incorrectos o estados inválidos, para garantizar la robustez del código.

Entre los **detalles a tener en cuenta para evitar errores** en el desarrollo de drivers se destacan:

- Asegurar consistencia en los tipos de datos (por ejemplo, uso de uint16_t o enumeraciones para identificar pines y puertos).
- Evitar dependencias circulares entre módulos (el driver no debe depender de la aplicación).
- Manejar adecuadamente condiciones de error o estados no válidos.
- Mantener una convención clara de nombres para funciones y variables.

Aplicaciones desarrolladas

App_2_1_Grupo5_2025

Autor: Plaate Ivan

Observaciones:
La modularización inicial permitió trasladar la lógica de encendido y apagado de los LEDs a una función (LED_Blink). El principal desafío fue entender cómo parametrizar el puerto y pin del LED, ya que HAL_GPIO_WritePin requiere estos valores específicos. Esto se resolvió pasando los parámetros desde el main.c a la función. Una recomendación es siempre usar constantes o enumeraciones para identificar LEDs, de modo que el código sea más legible y escalable.

App_2_2_Grupo5_2025

Autor: Torres Juan

Observaciones:
En esta aplicación el reto fue implementar la lectura del botón y la detección del flanco ascendente con antirrebote. Inicialmente, resultó confuso por qué debía usarse un static dentro de la función para recordar el estado anterior del botón; luego comprendí que esto permite que la función "recuerde" entre llamadas. También fue necesario entender el uso de punteros (&direction) para modificar una variable desde el driver. La recomendación es documentar claramente este mecanismo, ya que es un punto clave para principiantes en C y programación de microcontroladores.

App_2_3_Grupo5_2025

Autor: Angelillo Alessandro

Observaciones:
La complejidad aumentó al manejar 4 modos diferentes. El principal problema fue cómo organizar el código para que el main.c quedara limpio, y se resolvió dividiendo cada modo en una función independiente (APP3_Mode1, APP3_Mode2, etc.). También surgió la necesidad de pasar parámetros como el arreglo de tiempos para el Modo 3, lo que evitó tener valores fijos (“hardcodeados”) en el driver.

App_2_4_Grupo5_2025

Autor: Hualampa Leonel Javier

Observaciones:
El desafío principal fue implementar el cambio de velocidad con el botón, avanzando por distintos índices de tiempo. Para esto se separó la lógica en dos funciones: una que actualiza el índice y otra que ejecuta el parpadeo. El uso de sizeof(array)/sizeof(array[0]) fue útil para obtener la cantidad de elementos de forma genérica, aunque al principio resultó menos intuitivo que un length como en otros lenguajes.

Link del repositorio en Github con las Apps:
https://github.com/Alessandro-Angelillo/Grupo_5_TDII_2025/tree/main/AFP_2_TDII_2025