UTN Universidad Tecnológica Nacional

Facultad Regional Tucumán

Departamento Electrónica

Técnicas Digitales II

Actividad de Formación Práctica 3

Tema: Creación de drivers en STM32CubeIDE, programación de microcontorladores.

Profesor:

Ing. Rubén Darío Mansilla

ATTP:

Ing. Lucas Abdala

vencimiento:

25 de julio de 2025

Año: 2025

1. Objetivos

- Que el alumno desarrolle un ejemplo práctico y concreto de la creación de un driver, utilizando el *IDE* STM32CubeIDE, que se aplique a un módulo de la placa de desarrollo.
- Que el alumno logre experimentar un primer contacto con la utilización de drivers para el desarrollo de aplicaciones con el IDE que estamos utilizando.
- Que el alumno comprenda de la necesidad de la utilización de drivers, como una buena práctica de programación, al desarrollar aplicaciones para sistemas embebidos. Esto mejorará la portabilidad del código en caso de necesidad de cambiar de plataforma.
- Que el alumno se ejercite en las particularidades que presenta la utilización de drivers en la programación de microcontroladores.

2. Generalidades

Esta actividad de formación práctica es del tipo **práctica de laboratorio**.

Cómo desarrollar esta actividad:

- Esta actividad debe desarrollarse siguiendo las consignas del punto 3.
- Cada integrante del grupo debe tomar al menos una de las Apps desarrolladas en la actividad de formación práctica 1, desarrollar el driver GPIO y realizar las modificaciones correspondientes en el *main.c* de la App para que utilice las funciones del driver.
- Los integrantes del grupo deben ponerse de acuerdo para usar el mismo nombre para el driver GPIO y para las funciones que lo integran.
- Una vez modificadas todas las aplicaciones, se las ubicará en el repositorio grupal para su presentación. Todas las Apps deben presentar una coherencia en cuanto al formato del nombre del mismo y de las funciones desarrolladas para este.
- Se debe utilizar el repositorio grupal creado en GitHub para la presentación de las actividades de formación práctica:
  - Se debe crear una carpeta para cada actividad de formación práctica.
  - se deben ubicar en la carpeta correspondiente, en este repositorio, las aplicaciones desarrolladas para esta actividad de formación práctica respetando el siguiente formato de nombre: **App_ 1_ Y_ Grupo_ X_ 2025**.

  Nota sobre el formato de nombre de la carpeta: X es el número de su grupo y **Y** es el número de aplicación como figura en el enunciado.
- Se debe insertar el link al repositorio de GitHub al final del informe que se presenta con esta actividad de formación práctica.
- El archivo que contiene el desarrollo del informe de la actividad de formación práctica debe ser de tipo pdf y su nombre debe respetar el siguiente formato:

  AFP_ 3_ Grupo_ X_ TDII_ 2025.pdf

- Se debe realizar la presentación de las aplicaciones, en el laboratorio de Sistemas Digitales, funcionando según lo que pide la consigna, con su correspondiente defensa de lo desarrollado. Fecha a acordar. Cada alumno debe presentar y defender la aplicación que ha desarrollado para esta actividad.

Nota 1: Para el desarrollo de las aplicaciones de esta actividad de formación práctica se utilizará el entorno de desarrollo integrado STM32CubeIDE para aplicar sobre una placa de desarrollo STM32-NUCLEO-F4XX-YY.

Nota 2: Las placas de desarrollo sugeridas para las practicas son: STM32-NUCLEO-F401-RE, STM32-NUCLEO-F412-ZG, STM32-NUCLEO-F413-ZH ó STM32-NUCLEO-F429-ZI.

Nota 3: El desarrollo de las actividades de formación práctica se realizará y presentará de manera grupal.

3. Consignas para desarrollar

1. Tome como base una aplicación desarrollada en la actividad de formación práctica 1 y cree un driver para la utilización del módulo GPIO. Para hacer esto, siga los pasos detallados en la *Gu´ıa de creación de drivers* que está en un archivo en formato pdf, en el Ejemplo04, que se encuentra en el repositorio Tecnicas Digitales II, de aplicaciones de la materia.
2. Una vez creado el driver, haga las modificaciones necesarias en *main.c* para reemplazar las funciones de la HAL para el módulo GPIO, utilizadas originalmente, por las funciones creadas en el driver.
3. Compile, y realice el *degub* y la programación de la placa de desarrollo para verificar que la aplicación corre y ejecuta las operaciones correctamente como lo hac´ıa originalmente antes de la modificación.
4. Replique las modificaciones a todas las aplicaciones desarrolladas en la actividad de formación teórica 1.
5. Desarrolle un breve informe que contenga los siguientes Items:
   - 5.1. **Introducción al tema:** utilización de drivers en Apps desarrolladas para sistemas embebidos. En este ´ıtem explique las ventajas de desarrollar Apps utilizando drivers y cómo se realiza la creación y el desarrollo de un driver: pasos generales y detalles a tener en cuenta para evitar errores.
   - 5.2. **Aplicaciones desarrolladas:**
     - **Aplicación:** Nombre de la aplicación modificada.
     - **Autor de la modificación:** Apellido y nombres del alumno que realizó las modificaciones en cada aplicación.
     - **Observaciones:** sobre lo realizado en esa aplicación. En este campo puede compartir su experiencia en el desarrollo, los problemas que pudo haber enfrentado y como los solucionó. Recomendaciones si las hubiera.
   - 5.3. **Link al repositorio grupal:** en el que se encuentran las aplicaciones desarrolladas para esta actividad de formación práctica.

4. Bibliografía y documentación

- Documentación accesible desde el IDE en:
  Help → Information Center → STM32CubeIDE Manuals
- Documentación desarrollada por la cátedra: Creación de driver GPIO.pdf

ESTUDIANTE: Cusi Alejandro Daniel.....

ESTUDIANTE: Cancino Alan Maximiliano.....

FECHA DE INICIO: __/__/__ FECHA DE PRESENTACIÓN: 25/7/25

CONFORMIDAD DEL DOCENTE:.....

Informe: Utilización de Drivers en Aplicaciones para Sistemas Embebidos

En el desarrollo de software para sistemas embebidos, el uso de drivers modulares se ha vuelto una práctica fundamental para garantizar la escalabilidad, reutilización de código y claridad estructural. Un driver actúa como una capa intermedia entre el hardware y la lógica de aplicación, permitiendo desacoplar el manejo de periféricos (como leds, pulsadores, displays, etc.) del código principal del programa. Esta separación favorece el mantenimiento del código y la portabilidad entre distintos proyectos, placas o configuraciones.

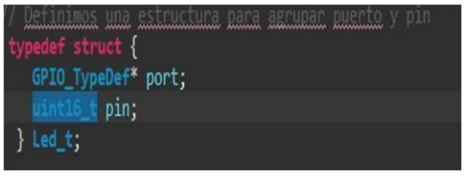
Crear un driver implica definir una interfaz clara y funcional para controlar un periférico específico. El proceso de desarrollo de un driver generalmente comienza con la creación de archivos separados (por convención, uno de tipo header .h y otro de tipo source .c) donde se declaran y luego se implementan funciones de inicialización, encendido/apagado, lectura o escritura, según el tipo de dispositivo a controlar. En nuestro caso particular, se diseñó un driver llamado API_GPIO para el manejo de salidas digitales (leds) y lectura de entradas (pulsador).

Algunos aspectos clave que se deben tener en cuenta durante su desarrollo incluyen:

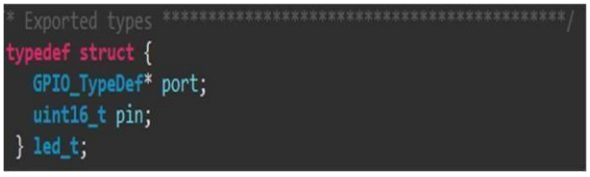
- Implementar el driver de manera genérica, es decir, que pueda ser utilizado por distintas aplicaciones sin necesidad de ser modificado.
- Mantener una correcta estructura de carpetas para seguir buenas prácticas de diseño.
- Verificar que las funciones del driver no dependan de valores codificados directamente (hardcodeados), sino que utilicen variables o #define bien definidos.

Aplicación desarrollada:
Nombre: App_1_1_Grupo_4_2025_Driver_GPIO
Autor de la modificación: Cruz Rodolfo Martin.

Observaciones:
Una vez acordado el driver a utilizar, denominado API_GPIO.h y API_GPIO.c, se procedió a modificar el archivo main.c. Durante este proceso, el IDE arrojó algunos errores. Uno de ellos fue olvidar eliminar (o comentar) la función static void MX_GPIO_Init(void) que ya no era necesaria, y que en nuestro caso se encontraba en la línea 74 de main.c. Otro inconveniente surgió por una confusión entre los archivos .h y .c del driver, al mezclar parte del código entre ambos. También tuve un conflicto con una variable definida en main.c sin modificar, que generaba interferencia con una definición similar dentro de API_GPIO.h. Esto se debía a diferencias en la escritura de la estructura



Captura del archivo main.c.
Se observa que Led_t; esta en mayúscula.



Captura del archivo API_GPIO.h
Se observa que led_t; está en minúscula.

Estos detalles causaron contratiempos, pero se solucionaron una vez identificados, corrigiendo el uso del tipo de dato led_t de forma coherente y ajustando el main.c al uso del driver elegido.

Otras cuestiones están relacionadas en como adaptar algunos ejemplos dados por el docente a nuestro main.c; las misma se muestra en la siguiente imagen:

```
#define LED1 LD1_Pin //GREEN LED
#define LED2 LD2_Pin //BLUE LED
#define LED3 LD3_Pin //RED LED
```

Captura de Ejemplo04_Driver_GPIO.

```
#define LED1  ((led_t){LD1_GPIO_Port, LD1_Pin}) // LED1 - Verde
#define LED2  ((led_t){LD2_GPIO_Port, LD2_Pin}) // LED2 - Azul
#define LED3  ((led_t){LD3_GPIO_Port, LD3_Pin}) // LED3 - Rojo
```

Captura del main.c grupal del App_

Para poder tomar este macro y adaptarlo a la primera app, consulte al Chatgpt. El cual brindo una estructura similar solo que tiene en cuenta los puertos como así también a los pin. Este cambio fue porque hay una estructura del siguiente tipo:

```
typedef struct {
   GPIO_TypeDef* port;
   uint16_t pin;
} led_t;
```

Recomendaciones:

- Consistencia en nombres de tipos y variables
- El error por no eliminar MX_GPIO_Init() en main.c se habría evitado con una revisión cruzada.

Aplicación desarrollada: App_1_2_Grupo_4_2025_Driver_GPIO

Autor de la modificación: Roqué Luciano J.

Observaciones:
En esta aplicación se buscó alternar la dirección de una secuencia de leds utilizando el pulsador onboard de la placa. El desafío principal fue adaptar el código ya funcional, escrito originalmente con llamadas directas a funciones HAL, para que pudiera utilizar el driver API_GPIO. Esto implicó reemplazar funciones como HAL_GPIO_TogglePin por funciones propias del driver, como toggleLed_GPIO(), y ajustar el manejo del pulsador con readButton_GPIO().

Uno de los aspectos más valiosos del trabajo fue comprender cómo abstraer la lógica de bajo nivel y encapsularla dentro de funciones reutilizables. Si bien al principio el código parecía funcionar correctamente sin driver, a medida que se fueron agregando funcionalidades y nuevas apps, se hizo evidente la ventaja de mantener una estructura modular.

Durante el desarrollo, surgieron errores menores relacionados con el uso incorrecto de tipos de datos o la ubicación de ciertas declaraciones, que fueron resueltos prestando atención a las convenciones de organización en los archivos .h y .c. También fue útil mantener la coherencia en los nombres y estructuras para evitar errores de compilación al migrar partes del código al driver.

Recomendaciones:

- Antes de comenzar con la adaptación de una app al uso de drivers, es importante que la app funcione correctamente con la HAL. Esto permite detectar más fácilmente si un error futuro proviene del driver o del programa principal.
- Usar nombres de funciones descriptivos dentro del driver mejora mucho la comprensión

cuando se lo reutiliza. Es recomendable mantener una estructura clara y ordenada del proyecto, separando bien los archivos del driver y los de la aplicación, y respetando las convenciones de nomenclatura.

Aplicación desarrollada: App_1_3_Grupo_4_2025_Driver_GPIO
Autor de la modificación: Cancino Alan Maximiliano

En esta aplicación desarrollamos una secuencia de control para tres LEDs, utilizando un pulsador onboard. La funcionalidad principal consiste en avanzar secuencialmente entre cuatro patrones distintos de parpadeo cada vez que se presiona el botón. Una vez alcanzado el cuarto patrón, la secuencia vuelve al primero, manteniendo así un ciclo continuo. Para la implementación inicial utilizamos la HAL de STM32, que facilitó el manejo tanto de los pines de salida como de la lectura del botón.

Uno de los principales desafíos fue garantizar transiciones limpias entre los distintos modos de parpadeo, evitando comportamientos erráticos o superposición de patrones. Esto se resolvió detectando correctamente los flancos descendentes del pulsador y estructurando el código de manera clara, con controles precisos del estado del sistema. Además, organizamos los LEDs en un vector para facilitar su manejo dentro de bucles y mantener un código más legible y escalable.

Una vez validado el funcionamiento con la HAL, adaptamos la aplicación para trabajar con un driver propio de manejo de GPIO. Esta modificación implicó reemplazar las llamadas directas a funciones HAL por funciones encapsuladas en el driver, logrando así una mayor separación entre la lógica de aplicación y el acceso directo al hardware. Esta abstracción no solo mejora la claridad del código, sino que también lo hace más reutilizable y mantenible para futuros desarrollos.

Observaciones: A modo de recomendación, considero clave empezar con una estructura lógica bien definida desde el comienzo, tanto en el diseño de la aplicación como en el desarrollo del driver. Personalmente, me resultó muy útil validar primero la lógica con la HAL y luego migrar al driver, ya que permitió detectar errores con mayor facilidad. También remarcó la importancia de manejar correctamente los estados en sistemas con pulsadores, ya que una detección inadecuada puede generar comportamientos impredecibles.

Aplicación desarrollada: App_1_4_Grupo_4_2025_Driver_GPIO

Autor de la modificación: Cusi Alejandro Daniel.

Observaciones:

En esta app desarrollamos los drivers GPIO para 4 secuencias de parpadeo, donde los 3 leds onboard parpadearan simultáneamente con ciclos de 100, 250, 500 y 1000 ms respectivamente, el cambio de secuencia vendrá dado por la pulsación del botón de usuario también onboard, luego de la última secuencia deberá volver al ciclo de la primera secuencia.

Se buscó mantener el orden e intentar un código entendible con uso de funciones compactas y utilizando comentarios. Si bien se uso la ayuda de IA, debido a varias pruebas fallidas, se intentó mantener el código lo máshumano posible. Esto fue mucho mas sencillo con la creación de las funciones Drivers_GPIO, que al ser creadas por el usuario presentan nombres mas familiares, pero su principal ventaja es permitirnos un código mas acotado y entendible.

Comentarios y Recomendaciones

- En mi caso hubo varios intentos fallidos debido a definir por duplicado. Por Ej., led_t ≠ Led_t donde, si bien no producía errores en la compilación, el código era erroneo y no desarrollaba la función buscada.

  Conocer el correcto llamado de pines de la placa a utilizar. Hubo varios errores de compilacion intentando utilizar el botón de usuario, buscando un poco en los archivos de la HAL de la placa a utilizar, decidí cambiar USER_Btn_Pin a GPIO_PIN_13 y la aplicación se desarrolló correctamente.

- Interiorizar en el manejo de funciones, sabiendo que tipo de variables toma y que es lo que devuelve.

Grupo 4 : Cancino Alan Maximiliano
Cruz Martín Rodolfo
Cusi Alejandro Daniel
Roqué Luciano Javier

Actividad de Formación Practica 3

Repositorio Grupal: Grupo-4-TDII-2025