UTN Universidad Tecnológica Nacional

Facultad Regional Tucumán

Departamento de Electrónica

Técnicas Digitales II

Actividad de Formación Practica 2

Tema: Creación de drivers en STM32CubeIDE,
Programación de microcontroladores

Profesor:
Ing. Rubén Darío Mansilla

ATTP:
Ing. Lucas Abdala

Grupo 8:
Gao Luciano, Mitre Emilio, Jorrat Tomas.

Vencimiento:
27 de septiembre de 2024

Registros de cambios

Revisión	Detalles de los cambios realizados	Fecha
0	Creación del documento	29 de agosto de 2024

Desarrollo:

1. Introducción:

Al desarrollar aplicaciones para sistemas embebidos, como en este caso donde se trabaja sobre microprocesadores ARM STM32, es importante o al menos muy recomendable tener en cuenta la portabilidad del código o programas. Si resulta necesario corregir, modificar o incluso aplicar un programa sobre otro hardware diferente, es mucho más eficiente realizar este trabajo si el código fue desarrollado de manera que se tuvo en cuenta la **escalabilidad** y **portabilidad**. Nos referimos a portabilidad al hecho de escribir drivers específicos para cada periférico, modulo, por ejemplo: SPI, I2C, UART, etc. Al tener drivers adaptados a cada uno de estos módulos se puede reutilizar el código en diferentes proyectos con diferentes microcontroladores, simplemente adaptando o reemplazando los drivers.

El uso de drivers brinda la posibilidad de **abstraer el hardware del código**, esto significa que el programa no necesita conocer los detalles de bajo nivel del hardware, como por ejemplo los registros específicos o configuraciones exactas del periférico. Al utilizar las API nos evitamos trabajar directamente con las funciones o manejo de registros de los módulos, periféricos u otras partes del hardware. A su vez los drivers permiten una **arquitectura modular** donde cada driver es independiente y el código puede ser reutilizado en otras aplicaciones o transferible a un sistema con hardware diferente, ya que el código del programa principal se vuelve más sencillo de **mantener** al usar funciones de las API y de ser necesario se modifican los drivers.

Trabajar con drivers en el desarrollo de programas embebidos proporciona numerosas ventajas, como la portabilidad, modularidad, el mantenimiento más sencillo y el aislamiento de errores. Al abstraer el hardware, se facilita la reutilización de código, lo que ahorra tiempo y recursos, especialmente cuando se trabaja en proyectos de largo plazo o escalables. Además, fomenta la colaboración y el trabajo en equipo, al permitir que diferentes desarrolladores trabajen de manera independiente en distintas capas del sistema.

Como guía usamos el siguiente archivo para la creación del driver GPIO, provisto por la catedra:

Creacion_Driver_GPIO.pdf

La creación del driver es la construcción de 2 archivos “.c” y “.h” vinculados uno con el otro a través de la instrucción “#include” y los “path” o recorridos que usa el IDE para compilar el programa, usando en este caso el módulo de propósito general I/O, conocido como GPIO. El contenido de estos archivos debe ser el que codifica la app STM32CubeMX por defecto para cada modulo, en nuestro caso usamos el GPIO. Dentro del archivo “.c” se debe declarar y desarrollar las funciones por el programador y dentro del archivo “.h” se debe declarar las funciones desarrolladas en el archivo .c

Orientado al desarrollo con STM32CubeIDE, los pasos generales a seguir para crear los drivers, en este caso para implementar las funciones que manejan el módulo GPIO son:

Generar el código para inicializar el hardware, como el reloj del sistema, módulos, periféricos. Lo cual podemos realizarlo utilizando la aplicación STM32CubeMX.

Luego debemos crear los archivos “API_X.c” y “API_X.h” para nuestra API y donde se desarrollará el código del driver. Los nombres de estos archivos pueden variar a gusto del desarrollador, pero es mejor si son descriptivos y se deben mantener las extensiones.

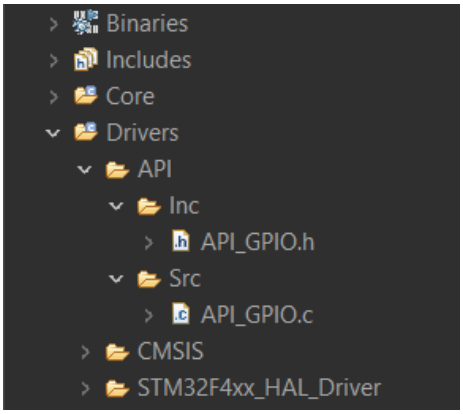

Figura 1. Explorador de proyectos.

Seguido de esto debemos asegurarnos de que ambos archivos estén dentro de la ruta de "includes", de lo contrario al momento de compilar tendremos errores. Para hacer esto debemos hacer click derecho sobre la carpeta API y luego elegir la opción "Add/remove include path"

Dentro del archivo "API_X.h" se encontrarán las declaraciones de las funciones, variables e inclusión de librerías.

Se trasladaron las declaraciones generadas al inicializar el código con el MX en "main.c" y es importante remover los prefijos "static" para poder utilizar las funciones en otros archivos de la aplicación. Además, se declararon las funciones propias del usuario para impletar en la API.

```c
#include <stdint.h>
#include <stdbool.h>

#ifndef API_INC_API_GPIO_H_
#define API_INC_API_GPIO_H_

typedef uint16_t led_t;
typedef bool buttonStatus_t;


void SystemClock_Config(void);
void MX_GPIO_Init(void);
void MX_USART3_UART_Init(void);
void MX_USB_OTG_FS_PCD_Init(void);

void writeLedOn_GPIO(led_t LDx);
void writeLedOff_GPIO(led_t LDx);
void toggleLed_GPIO(led_t LDx);
buttonStatus_t readButton_GPIO(void);
```

Figura 2. archivo API_X.h

En el archivo "API_X.c" se realizará el desarrollo de las funciones e inicializaciones que se declararon en el archivo ".h". Es importante realizar los "#include" de las librerias correspondientes y principalmente la de nuestro archivo "API_X.h".

Nuevamente se debe trasladar el código generador por el MX desde "main.c" para la inicialización de los módulos y luego se implementará el desarrollo de las funciones propias del

usuario para utilizar con mayor facilidad el hardware del sistema, como por ejemplo las funciones de la HAL que permiten cambiar el estado de los pines en el módulo GPIO.

```
void writeLedOn_GPIO(led_t LDx){
    HAL_GPIO_WritePin(GPIOB, LDx, GPIO_PIN_SET);
}

/*
 * @brief Apagar LED GPIO
 * @param led_t LDx
 * @retval ninguno
 */

void writeLedOff_GPIO(led_t LDx){
    HAL_GPIO_WritePin(GPIOB, LDx, GPIO_PIN_RESET);
}

/*
 * @brief Alternar LED GPIO
 * @param led_t LDx
 * @retval ninguno
 */

void toggleLed_GPIO(led_t LDx){
    HAL_GPIO_TogglePin(GPIOB, LDx);
}

buttonStatus_t readButton_GPIO(void){
    return HAL_GPIO_ReadPin(GPIOC, USER_Btn_Pin);
}
```

Figura 3. archivo API_X.c

Una vez creado nuestro Driver, es importante realizar el “#include” en programa principal para poder utilizar las funciones de la API.

Otros detalles para tener en cuenta son las variables que esperan recibir y retornan las funciones, ya que al crearlas en la API o implementarlas en el programa principal es fácil cometer errores. Lo mismo si se crea un tipo de variable del usuario, las cuales se acostumbra a terminar en “_t” debemos tener claro a que tipo de variables o estructura hacen referencia.

2. Aplicaciones:

- **App 1.1:** Esta aplicación consiste en el encendido y apagado de manera secuencial de los tres leds integrados de la placa de desarrollo. La secuencia debe encender 200 ms y apagar 200 ms cada led comenzando por el LED1 (Green), continuando con el LED2 (Blue) y luego el LED3 (Red) para volver a iniciar con el LED1.

  Modificaciones: Las principales modificaciones que se realizaron sobre el programa principal al implementar los drivers fue el remplazo de las funciones de la HAL, excepto la función HAL_Delay. Se declaro un vector para seleccionar cada uno de los Leds integrados en la placa y con una función “for” para recorrer el vector y encenderlos según la secuencia requerida.

- **App 1.2:** Aplicación para alternar entre 2 secuencias de Leds al pulsar el botón integrado en la placa de desarrollo.
  Secuencia Normal, enciende y apaga los leds cada 200ms en orden Verde, Azul y Rojo
  Secuencia Invertida, cambia el orden de parpadeo a Rojo, Azul y Verde

  Modificaciones:

- Se definió una variable N, la cual indicara el tamaño del vector de Leds haciendo más fácil añadir otros leds de ser necesario.
- Se definió una enumeración para luego poder facilitar la selección de secuencias.

```
typedef enum{
    secuencia_normal=1,
    secuencia_invertida=-1
} secuencia;
```

- Al igual que en la primera aplicación se remplazaron las funciones de la HAL para implementar las creadas en nuestro Driver.

- **App 1.3:** Aplicación para alternar entre 4 secuencias de parpadeo de leds al pulsar el botón integrado en la placa de desarrollo.
  Secuencia 1: Parpadeo cada 150ms en orden Verde, Azul y Rojo.
  Secuencia 2: Parpadean todos los leds de manera simultánea cada 300ms.
  Secuencia 3: LED1 (Verde) parpadea cada 100ms, LED2 (Azul) cada 300ms y LED3 (Rojo) cada 600ms.
  Secuencia 4: Parpadean de manera simultánea el LED1 y LED3 cada 150ms, cuando esta apagados los LED1 y 3 se enciende el LED2.

  Modificaciones:

  - Se mantuvo la misma estructura de la App 1.2, se agregaron las cuatro secuencias a la enumeración (de manera que les corresponden los valores 1,2,3 y 4).

```
typedef enum{
    secuencia_1=1,
    secuencia_2=2,
    secuencia_3=3,
    secuencia_4=4
} secuencia;
```

  - Se modificaron las condiciones para poder repetir el ciclo de secuencias una vez se llegó a la secuencia 4.
  - Se remplazo en las funciones correspondientes a cada secuencia las funciones de la HAL por las funciones creadas en nuestro Driver.

- **App 1.4:** Aplicación para cambiar la frecuencia de parpadeo de los leds, se encienden y apagan todos los leds al mismo tiempo. Al presionar el botón integrado se alterna entre los tiempos 100, 250, 500 y 1000 ms.

  Modificaciones:

  - Se utilizo la misma estructura general de la aplicación 1.3 pero modificando las funciones “secuencia” (1,2,3 y 4).
  - En cada “secuencia” se utiliza la función de nuestro driver toggleLed_GPIO para alternar el estado de los leds y la se modifica la frecuencia de encendido con la función HAL_Delay.

3. Repositorio Grupal:

github.com/Grupo_8_TDII_2024